User Guide - English

WebTransactions V7.5

Concepts and Functions

Edition August 2010

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

Contents

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 scaleable 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 WebTransactions supply units

The following table provides an overview of the supply units and platforms for which WebTransactions is currently available:

Supply unit	WebTransactions platform	Supported protocol
WebTransactions for OSD	BS2000/OSD Solaris Linux Windows	9750
WebTransactions for MVS	Solaris Linux Windows	3270
WebTransactions for openUTM	BS2000/OSD Solaris Linux Windows	UPIC

Table 1: WebTransactions supply units

Each variant consists of the WebTransactions runtime system together with a special host adapter via which the communication with the host application is carried out, and a host adapter for dynamic Web content. Each supply unit contains the WebLab development environment which you use to link a host application to the WWW and optically prepare the host formats and also extend their functionality.

Due to the modular architecture of WebTransactions and its compliance with open standards, new host adapters for additional application types can be implemented quickly and cost-effectively on request - even within project solutions.

1.2.1 Application-specific host adapters

The host applications usually communicate using a proprietary protocol, which is where the application-specific host adapters and the individual supply units get their names from.

WebTransactions for openUTM

WebTransactions for openUTM is used to link openUTM host applications which use FHS or FORMANT formats or operate via the UPIC client server interface, to the Web. WebTransactions for openUTM is available on the WebTransactions system platforms BS2000/OSD, Solaris, Linux and Windows.

With Solaris, Linux and Windows, you can use any Web server you wish. Under BS2000/OSD, WebTransactions requires the Apache Web server.

The openUTM host application can base on any of these platforms. The host adapter for communication with the host application is based on the openUTM client (UPIC), or in other words, WebTransactions and the host application can be run on different platforms.

The BS2000 program `IFG2FLD` is delivered with WebTransactions for openUTM. This is used to convert the descriptions of the host formats from the IFG library into format source descriptions. You can then use WebLab to automatically generate templates from these format source descriptions. These generated templates then form the basis for the individual layouts of the various formats.

WebTransactions for OSD

WebTransactions for OSD is used to link any BS2000/OSD dialog application (TIAM, DCAM) to the Web. These applications may also be openUTM applications.

WebTransactions for OSD is available on the system platforms BS2000/OSD, Solaris, Linux and Windows and operates there with any type of Web server. Under BS2000/OSD, WebTransactions requires the Apache Web server.

The host adapter used to communicate with the host application is based on a 9750 emulation. WebTransactions can communicate with the host application in the same way as a 9750 terminal. A so-called Automask template is used to convert the 9750 data stream 1:1 into HTML. The most frequently used keys of a 9750 terminal, such as the DUE, K and F keys, are shown in the form of buttons in the browser, all the other special keys and programmable keys in form of a list or in form of buttons too. With current browsers, you can also control the host application via the keyboard.

You have the following options for further development of the Web interface:

- You can globally edit the entire interface by modifying the Automask template.
- You can edit the formats individually using a special capture mechanism. This process records “snapshots” of the formats which are then converted to the relevant templates. These form the basis for individual editing.
- In order to ensure that certain specified areas of the format have the same layout, for example, the button bar, you can take a master template as a basis when generating Automask and format-specific templates.

The 9750 emulation is integrated into WebTransactions for OSD.

The BS2000 program `IFG2FLD` is delivered with WebTransactions for OSD. This is used to convert the descriptions of the host formats from the IFG library into format source descriptions. You can then use WebLab to automatically generate templates from these format source descriptions. These generated templates then form the basis for the individual layouts of the various formats.

WebTransactions for MVS

WebTransactions for MVS is used to link any MVS or z/OS dialog application to the Web.

WebTransactions for MVS is available on the WebTransactions system platforms Solaris, Linux, Windows and operates there with any type of Web server.

The host adapter used to communicate with the host application is based on a 3270 emulation. WebTransactions can communicate with the host application in the same way as a 3270 terminal. The so-called Automask template is used to convert the 3270 data stream 1:1 into HTML. Special keys on the 3270 terminal are shown as buttons in the browser or are given in a list. With current browsers, you can also control the host application via the keyboard.

As in WebTransactions for OSD you have a range of development options for the Web interface:

- You can globally edit the entire interface by modifying the Automask template.
- You can edit the formats individually using a special capture mechanism. This process records “snapshots” of the formats which are then converted to the relevant templates. These form the basis for individual editing.
- In order to ensure that certain specified areas of the format have the same layout, for example, the button bar, you can take a master template as a basis when generating Automask and format-specific templates.

The 3270 emulation is integrated into WebTransactions for MVS.

1.2.2 Host adapter for dynamic Web content

WebTransactions for dynamic Web content is used to access any Web content using the HTTP protocol and is delivered as additional host adapter with all supply units.

WebTransactions for dynamic Web content is available for the WebTransactions system platforms BS2000/OSD, Solaris, Linux and Windows and can access any Web server via the HTTP protocol.

The host adapter via which communication with the Web application is carried out is based on the HTTP protocol. WebTransactions supports the encryption of messages via HTTPS as well as client and server authentication via certificates. In this way, WebTransactions can communicate with the Web application in the same way as a browser. You have the option of forwarding the data received from the Web application (HTML or XML) either 1:1 to the browser or modifying the data, for example, to use only certain parts of the data, to regroup the data or to add additional data.

WebTransactions for dynamic Web content supports the access to Web services. By importing the Web service descriptions in WSDL (**W**eb **S**ervice **D**escription **L**anguage), a simple and transparent mechanism for utilising remote services via SOAP (**S**imple **O**bject **A**ccess **P**rotocol) is available.

1.3 WebTransactions documentation

The WebTransactions documentation consists of the following documents:

- A Reference Manual which also applies to all supply units and which describes the WebTransactions template language WTML. This manual describes the following:

  Template Language

  After an overview of WTML, information is provided about:

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

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

  Connection to openUTM applications via UPIC

  Connection to OSD applications

  Connection to MVS applications

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

  – The installation of WebTransactions with each type of host adapter.

  – The setup and starting of a WebTransactions application.

  – The conversion templates for the dynamic conversion of formats on the web browser interface.

  – The editing of templates.

  – The control of communications between WebTransactions and the host applications via various system object attributes.

  – The handling of asynchronous messages and the print functions of WebTransactions.

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

  Access to Dynamic Web Contents

  This manual describes:

  – How you can use WebTransactions to access a HTTP server and use its resources.

  – The integration of SOAP (Simple Object Access Protocol) protocols in WebTransactions and the connection of web services via SOAP.

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

  Client APIs for WebTransactions

  This manual describes:

  – The concept of the client-server interface in WebTransactions.

  – The `WT_RPC` class and the `WT_REMOTE` interface. An object of the `WT_RPC` class represents a connection to a remote WebTransactions application which is run on the server side via the `WT_REMOTE` interface.

  – The Java package `com.siemens.webta` for communication with WebTransactions supplied with the product.

- 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 WebTransactions manual “Concepts and Functions” provides an introduction to working with WebTransactions and is intended for anyone not yet familiar with the product.

Rather than going into the syntactic details of individual statements or the specifics of the interface, this manual explains all the underlying concepts of WebTransactions which are the same for all the supply units. Equipped with this information, you will easily be able to find your way around the suite of WebTransactions manuals.

Chapter 2 provides a brief introduction to the features and mode of operation of WebTransactions. Chapter 3 contains the most important WebTransactions definitions including a detailed description of the object concept. Chapter 4 concentrates on the dynamic sequencing of a WebTransactions session. Chapter 5 describes how you administer WebTransactions. Chapter 6 presents WebLab, the WebTransactions development environment and describes its most important functions.

The detailed lists at the back of the manual - the abbreviations, glossary, list of related publications, and the index - allow you to quickly locate specific information in the manual.

Scope of this description

This documentation is valid for all supply units, irrespective of the WebTransactions platform on which they are running.

1.5 New features

This section provides an overview of the most important new features in WebTransactions Version 7.5 and indicates where you can find a more detailed description.

Type of new feature	Description
Global system object attributes: – New system object attribute LT_REPLACE_STRING – New system object attribute PREVENT_EXIT_SESSION	– page 78 – page 79
Global functions: – New function moveFile() – New function copyFile() – New function isRequestWaiting() – New parameter all in listFolder()	WebTransactions manual “Template Language”
Built-in classes and methods: – New method String.fromCharCode – New method string.charCodeAt – New method WT_Filter.dataObjectToFormattedXML – Change for the output of the toString() method on objects and arrays: better serialization/deserialization	WebTransactions manual “Template Language”
Exceptions: New attributes strLine, strColumn and strText at the exception object	WebTransactions manual “Template Language”
C/C++ user exits: New argument SendMail	WebTransactions manual “Template Language”
WebTransactions for openUTM: – New host data object attribute Unicode – New attribute Unicode at WT_HOST_MESSAGE	WebTransactions manual „Connection to openUTM Applications via UPIC“
WebTransactions for OSD: – New value for the system object attribute HOST_CHARSET: 9763-UNICODE – New value for the system object attribute TERMINAL_TYPE: UTF-8 – Change for the system object attributes END_MARK und LZE_CHAR	WebTransactions manual „Connection to OSD Applications“

Type of new feature	Description
WebTransactions for Dynamic Web Contents: – New system object attribute `COMMUNICATION_FILE_NAME` – New system object attribute `COMMUNICATION_FILE_TYPE` – New system object attribute `METHOD`	WebTransactions manual „Access to Dynamic Web Contents“
Web Frontend for Web Services: The support of business objects is omitted.	

1.6 General solutions

A number of generally applicable solutions are provided free of charge with WebTransactions Version 7.5. Examples are:

The connection to the SAP Enterprise Portal
A SAP-certified business package enables the seamless integration of WebTransactions applications into the `mySAP` enterprise portal.

The connection to portals that support JSR168
A portlet that is compliant with the JSR168 permits the seamless integration of WebTransactions applications in the corresponding portal.

A mobile portal with WebTransactions
You can quickly create a personalised portal for your host applications with a WTBean. You receive a portal interface designed for PCs and, in parallel, an interface for PDA (**P**ersonal **D**igital **A**ssistant, e.g. Pocket LOOX), which is optimized for mobile access.

The BS2000 console connection
Access to the BS2000 console enables the administration of the BS2000 via a browser or a PDA (e.g. Pocket LOOX).

For further information on special solutions refer to the WebTransactions Homepage (*ts.fujitsu.com/products/software/openseas/webtransactions.html*). There you can also download the solutions.

1.7 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
italic font	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
{*alternative1* | *alternative2* }	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
ℹ	Important notes and further information
►	Prompt telling you to do something.
⇨	Refers to detailed information

2 Functional overview

This chapter describes the different fields of application of WebTransactions - from automatic 1:1 conversion to the integration of several host applications as well as the main functions of WebTransactions and how to use them (see also section “WebTransactions function range” on page 29).

2.1 Possible applications

The Internet has long since established itself as the decisive networking technology for data processing, both commercial and non-commercial. Analysts from the leading management consultants now believe that only those companies who use Internet technology to handle their business procedures will be able to compete successfully in the market. The main reason for this development is the fact that Internet technology not only provides access to the global “electronic marketplace” through its World Wide Web graphical interface, but the same technology can be used to process internal or external procedures within an intranet or extranet.

Most companies today have a heterogeneous system environment in which business-related applications and data are distributed as required over PC servers, departmental systems, or mainframes. Standard software and customized solutions coexist in the same way as the wide variety of operating systems and databases.

To allow for the smooth handling of business processes, all applications and data must be available immediately wherever they are required. In the context of the further development and optimization of processes, these applications and data must also be intercompatible.

To achieve your objectives, you will require software compatible with the existing system environment, which enables you to integrate those applications in which you have invested heavily into the Internet/Intranet without modification.

WebTransactions is a product which has been tried and tested with great success in various sectors and scenarios. It comprises the following distinct modules:

Automatic 1:1 conversion (browser-based emulation)

WebTransactions provides converters for converting proprietary mask formats into browser-compatible HTML documents. These automatically generated HTML pages have the same Look & Feel as the terminal.

It is possible the access host applications and other data sources via any Web browser instead of through terminals or PCs with terminal emulations. This access can take place enterprise-wide via the Intranet, or even worldwide via the Internet. The system and data access control mechanisms of your host applications are obviously also available when accessing via the Web. Furthermore, Web access can also be offered parallel to terminal access.

Problems in the distribution of client software do not occur, as Web browsers are universally available and are not restricted to any particular platform.

➪ For more detailed information, please refer to the manual for the host adapter in question.

Designing the interface (GUIfication)

Based on the result of the 1:1 conversion, you can edit the design of the Web interface. All HTML and JavaScript options are supported.

You can thus give your trusty host applications a whole new look, and adapt them to your company's corporate design. It is also possible to integrate an online help system. This makes your host applications more attractive to new users and customer groups, who are used to working with a PC and find the layout of alphanumeric mask systems both complicated and old-fashioned.

On the other hand, IT experts who have been working with your host applications for years, and who are used to the minimalist but effective tools, may feel the overhead of a user guidance system with helpful graphical effects is too much of a good thing. WebTransactions allows you to provide the same service to various user groups via different interfaces.

➪ For more detailed information, please refer to section "Editing templates" on page 170 and the manual for the host adapter in question.

Designing dialog sequences (interface reengineering)

But WebTransactions offers much more than a “face lift” for your host applications, allowing you to redesign your dialog sequences. The rigid 1:1 assignment between an HTML page and host format is now obsolete: with WebTransactions you can actively modify the dialog strategies provided by the host applications, filter out or add input/output elements, and combine or expand individual dialog steps.

You can thus design the sequences of the host applications to be more user-friendly, thereby increasing acceptance and the efficiency of online dialogs.

➪ For more detailed information, please refer to the sections “Dialog between WebTransactions and the host application” on page 108 and “Dialog between WebTransactions and the browser” on page 110.

Integrating applications (business process reengineering)

The laws of market dynamics force business processes to be highly flexible. The IT infrastructure is faced with the task of providing optimum support for any changes to your environment. However, due to the complexity of heterogeneous IT systems which have developed over years, difficulties frequently arise when you try to produce flexible, efficient adaptations: for each modification, you have to wade through a mire of mutual references and dependencies. This increases your costs with results that are rarely satisfactory. The integration of existing heterogeneous systems into a uniform, transparent IT concept is therefore a key requirement in modern information technology.

WebTransactions facilitates such an integration. Different host applications on different host platforms can be integrated into a single WebTransactions application on the Web server and provided with a common Web interface. Existing host applications are accepted just as they are, they are given a common web interface. You do not have to modify the application logic. The host applications are addressed by WebTransactions via their existing proven interfaces.

The integration function is provided at a clearly defined, central location in the templates and configuration files of WebTransactions.

The end user at the Web browser is generally not aware of which host application provides the requested services, or of the host platform on which the application runs. When working with the WebTransactions application, the complexity of the underlying IT infrastructure remains hidden.

➪ For more detailed information, please refer to section “Starting a WebTransactions dialog application” on page 92 and the manual for the host adapter in question.

Portals

In many companies, various types of information are made available in portals to ensure that, to the greatest possible extent, the information can be accessed independently of time and location. A key question when constructing a portal is whether all the relevant applications and data can be accessed from the portal. For the mySAP Enterprise Portal, the Oracle portal and all portals that support JSR168 and WSRP, WebTransactions provides out-of-the-box integration for BS2000/OSD and z/OS applications.

At the same time, you can also use WebTransactions to build your own portal. A supplied GUI module provides role-specific portal GUIs. The fully functional example installation that is supplied with the product allows portal users to define and use customized access to their applications within the portal quickly and easily without any prior knowledge. This means that a small-scale portal solution containing all the important accesses to applications and data can be created quickly and efficiently. In addition, the WebTransactions portal provides a mode of operation that is optimized for PDAs (e.g. the Pocket Loox) and which offers full roaming capability.

➪ For notes on WebTransactions applications created for portals, see section “Designing templates for portal use” on page 177.

Remote applications

XML offers you a basis for further integration. Several remote WebTransactions applications can be combined to create a network-wide global solution. Data from the participating WebTransactions applications is exchanged in the form of XML documents using the HTTP or HTTPS protocols.

One obvious use could be as follows. The branches of a company process locally obtained data using WebTransactions branch applications. The data resources which are to be used centrally on an inter-branch basis can be combined according to defined criteria within the framework of a web integration solution, and can then be made available for access via a common web interface. Here, the use of cooperating WebTransactions instances, ensures that the load on the network is kept to a minimum.

➪ For more information, refer to the WebTransactions manual “Access to Dynamic Web Contents”.

Using client programs

Often in environments with a range of different procedural policies, new IT solutions are used in parallel with the old, tried and tested solutions. The new WebTransactions client concept can be used to combine old and new solutions. The keyword for this is `WT_REMOTE`. These open interfaces are used to enable various programs to access the resources of WebTransactions applications (objects and methods) and to make use of their functionality. Any application can send a query in the form of a multi-part HTTP message in a particular format to the web server. The server forwards the message to the `WT_REMOTE` interface of WebTransactions, where it is then interpreted and processed.

The main difference between this procedure and using a browser as the default client of a WebTransactions application is, that it is not only possible to access the HTML page of a WebTransactions application, it is also possible to directly access the remote application using a type of remote procedure call. So, for the first time ever, it is now possible for a client to have active influence over the WebTransactions application, to remotely control this application, or to use the functionality of a WebTransactions application for its own purposes.

For more detailed information, refer to the WebTransactions manual “Client APIs for WebTransactions”.

Load distribution using clusters

The number of users and frequency of use of the Internet and especially of intranets is steadily increasing. It is not unusual that, a single integration server, no matter how powerful it is, is no longer in a position to guarantee all end users the necessary short response times or to prevent system unavailability caused by overloading.

In terms of performance and reliability, WebTransactions has the perfect answer. Several integration servers can be combined to form a so-called cluster. These integration servers do not all need to be of the same type. All of the integration platforms supported by WebTransactions, such as Windows, OSD, Solaris and LInux, can be combined as required. The cluster is defined and set up via the administration and development environment.

One or more of the servers in the cluster takes on the role of the cluster controller on which the cluster definition is stored. The other integration servers, also known as cluster members, each have a copy of the WebTransactions application or templates.

At runtime, the controller distributes the total load to the individual integration servers of the cluster in accordance with the specified strategy. You can choose from a variety of different distribution strategies, from a simple “round robin” strategy through to a “load balancing” strategy which takes into account the current load on the individual machines. As far as the end user is concerned, apart from the shorter response times, there is no difference between the a single server configuration and the cluster configuration. As a result, WebTransactions-based integration solutions are easily scalable.

For more detailed information, refer to section “Cluster concept” on page 142.

Blade server support

The Blade Server Systems by Fujitsu Technology Solutions have a high level of flexibility, scalability and availability and are easy to service. Compared to conventional servers, they provide a high computing performance per space unit with very low energy consumption. Using the optional deployment software, you can construct an automated multi-server installation.

WebTransactions is easily installed on the blades of a blade server and is readily available on many blades using an image. The creation and use of WebTransactions applications on blade servers are the same as with any standard server.

The creation and administration of a WebTransactions cluster on a blade server is as simple as the individual installation. A possible use for a cluster here would be, for example, on an application operated with many clients where it is necessary to distribute the load over several blades.

For more information, please refer to section “WebTransactions on a blade server” on page 149.

Application Mobility

With WebTransactions, you can access your host application’s data at any time and from any place. Thanks to the automask for small displays, business-critical applications for BS2000/OSD or z/OS can be made available to PDAs without difficulty.

Thanks to the easy-to-use session roaming capabilities, it is also possible to transfer access to a session between different client devices or continue working seamlessly after a temporary loss of connection.

2.2 WebTransactions function range

This section describes the functions available in all WebTransactions supply units.

WebLab development environment

The WebLab development environment is the tool which allows you to link your host application simply and quickly to the Web. It is immaterial whether your WebTransactions server is an Windows system, a Unix system, or a BS2000 system, since WebLab can create and manage WebTransactions applications both locally and remotely. Data is transferred from the server to your PC using the HTTP or HTTPS protocol and is returned via the same route.

WebLab offers a particularly user-friendly facility for the individual programming and testing of templates which allows you to edit your templates via the network. Parallel to the editing session, WebLab manages a WebTransactions session in a Web browser so that you can view the effects of your changes instantaneously.

WebLab offers direct access to WebTransactions objects, and allows you to insert WTML tags in your template within a dialog. For convenience, it provides a series of wizards which simplify those tasks identified from experience as frequently required when integrating your host application into the intranet/Internet.

A more detailed introduction to using the WebLab functionality can be found in chapter "The WebLab development environment" on page 155 and in the introductory chapter of the WebLab online help system. There you will also find a few practical examples which explain the workings of WebLab.

WTML templates

WTML templates (or simply templates for short) form the core of each WebTransactions application. They are automatically generated by WebLab from the format definitions of the host applications and - if desired - can be programmed individually using HTML and WTML language resources. At runtime, they are evaluated by WebTransactions and define the layout of the HTML pages displayed in the browser, control the WebTransactions application, and define the steps for communicating with the host applications.

In WebTransactions for openUTM, a corresponding template is automatically generated for each format. The OSD and MVS supply units include an Automask template, which is responsible for the conversion of all screen formats by default. An option is also available here for generating individual templates (capture mechanism).

➪ For more detailed information, refer to chapter “What is a WebTransactions application” on page 39 and the manual for the host adapter in question.

Master template for template generation

WebTransactions uses master templates for generating the Automask and format-specific templates. This gives you a uniform layout. The biggest advantage of the master template concept can be seen with host applications which use a large number of formats with similar structures: e.g. fixed header, working area and footer formats.

In such a situation, it is enough to specify the structure once in a master template and assign this as the template when generating both format-specific templates and Automask templates. All templates generated as a result of this procedure will automatically have the desired structure.

➪ Further information on possible uses can be found in section “Master, class and module templates” on page 51 and in the manual for the host adapter in question.

Template language WTML for individual programming

Based on the templates generated, it is possible to design the graphical layout of the Web interface and - if desired - restructure the dialog sequences.

The template language WTML (WebTransactions Markup Language) is provided for the individual programming of templates. WTML consists of two components: on the one hand, the WTML tags with which you embed the WebTransactions-specific section in HTML pages and, on the other, WTScript, a server-side script language with which, for example, you can control host applications and process host application data.

WTML therefore allows you not only to modify the appearance of the GUI but also to design the processing logic to meet your own specific requirements: with WTML, you can actively control the dialog with the host application or integrate multiple host applications within a single web GUI.

WTML tags are very similar to HTML. In WTScripts, you can also use a variety of language resources similar to JavaScript for programming server-side processing steps. This means that you can use the same language resources and the same language for the template logic and for programming the presentation logic on the client side.

HTML and client-side JavaScript are not part of WTML, but are transferred to the browser unchanged. WebTransactions is thus open to future HTML and JavaScript extensions.

The template technique is so flexible that you can even implement any dynamic HTML structures without an underlying host application.

For more detailed information, refer to section “WTML templates” on page 48 and the WebTransactions manual “Template Language”.

Support for XML

WebTransactions provides a number of easy-to-use ways of analyzing XML data. In this way, you can integrate resources that are present in XML into your WebTransactions applications quickly and easily.

It is also possible to generate XML from internal data structures. This means, for example, that you can save internal states and read these when you restart a WebTransactions session.

Support for Unicode

The application-specific host adapters WebTransactions for OSD and WebTransactions for openUTM support Unicode. They create data encoded in UTF-8, which is passed straight through to the browser and back.

The following WebTransactions components and application-specific host adapters do not support Unicode:

- the WebTransactions kernel

  The interpreter for the template language WTML (WebTransactions Markup Language and WTScript) does not support Unicode:
  - WTScript templates are still interpreted as ISO-8859 characters.
  - Each byte of a string will still be interpreted as one character.

- WebTransactions for MVS

- WebLab (Unicode characters cannot be represented)

➪ Further information on the Unicode support can be found in the WebTransactions-manuals „Connection to OSD Applications“ and „Connection to openUTM Applications via UPIC“.

WTBeans as reusable components

WebTransactions provides you with reusable components, known as WTBeans, for template programming. WTBeans correspond to a full WTML document or a part of such a document and a distinction is consequently made between standalone and inline WTBeans.
WTBeans support both the display of data in the browser and communications with the host applications.

Every WTBean possesses a description of its properties. WebLab uses these properties to generate a graphical user interface which enables you to edit the WTBean’s properties.

➪ Further information on possible uses can be found in section “WTBeans” on page 53.

Support for various interface styles and languages

A dialog interface can be assigned various template sequences (interface styles), which can be selected as desired at the browser. For instance, a host application may be offered in its previous format for users who are used to this, while also being available in variants containing lavish graphics. In addition to various interface styles, it is also possible to use different languages in parallel.

All interface styles are managed centrally on the WebTransactions host, and after modification are immediately available to all clients (Web browsers).

For more detailed information, refer to section “Subdirectories for style and language variants” on page 56.

Simple concept for integrating several host applications

Based on its concept of communication objects, WebTransactions offers a simple, transparent option for integrating different host applications under a common Web interface. A separate communication object is created for each host connection. All connection-specific data - be it control information or exchanged user data - is stored in this object. From the programmer’s viewpoint, this simplifies and clarifies the handling of several parallel open connections to different hosts. End users at the Web browser are usually completely unaware of the fact that they are working with different hosts.

For more detailed information, refer to section “Host communication object WT_HOST.Comobj - managing a host connection” on page 85 or the manual for the host adapter in question.

Security functions

The security mechanisms of the host applications (system access control, data access control, restart) are obviously also available when accessing via WebTransactions. In addition, you can restrict system access to the WebTransactions CGI programs using the security mechanisms of the WebServer software or the file system access rights.

WebTransactions also offers its own security and user concept which allows you regulate the access to WebTransactions applications for both developers and administrators. Edit and administration rights can be defined individually using special configuration files.

The administration program is an autonomous WebTransactions application which is closely connected to WebLab in order to ensure the required access controls.

For more detailed information, refer to chapter “WebTransactions server” on page 131.

Session management

WebTransactions is responsible for managing active sessions. It records the client from which a request originates, and the host application with which the user wishes to work.

The WebTransactions administration program can be used to obtain an overview of the sessions currently active, output the temporary files of individual sessions (e.g. trace files), terminate sessions, and enable or disable the entire WebTransactions application. Administration could not be easier and takes place via a separate WebTransactions application.

For more detailed information on the administration of the WebTransactions application, refer to section “Administering a WebTransactions application” on page 127.

Java integration

The Java integration offered by WebTransactions allows you to extend the scope of your WebTransactions application. In combination with WTScript, the JavaScript-based script language used by WebTransactions, Java integration offers you the following possibilities:

- Generation and use of Java objects and Java arrays
- Java method calls
- Reading and writing of Java attributes
- Use of WTScript operators on Java objects
- Java-based exception handling

If Java programs or applications are to run on the WebTransactions system then a Java runtime environment (Java Virtual Machine, JVM) must be available on this system. Separate Java runtime environments are available for the various WebTransactions integration platforms (Windows, Solaris, Linux, BS2000/OSD).

For more detailed information, refer to the WebTransactions manual “Template Language”.

Connecting of Web services via the SOAP protocol

WebTransactions allows you to connect to Web services via the **S**imple **O**bject **A**ccess **P**rotocol (SOAP). The integration of the SOAP protocol in WebTransactions is based on the WebTransactions HTTP host adapter as well as on the definitions of the SOAP and WSDL (**W**eb **S**ervices **D**escription **L**anguage) protocols. WSDL provides XML language rules for describing the capabilities of Web services.

The XML-based SOAP protocol provides a simple, transparent mechanism for the exchange of structured, typed information between systems within a decentralized, distributed environment.

SOAP provides a modular package model together with mechanisms for data encryption within modules. This permits a simple description of the external interfaces for a remote application that can be accessed in a web (web service).

For more detailed information, refer to the WebTransactions manual “Access to Dynamic Web Contents”.

Using user-defined C/C++ routines (user exits)

User-defined C/C++ routines can be integrated as user exits into WebTransactions applications, thereby extending the functionality of the host applications. It is also possible to use several user exit libraries.

With user exits, you can create additional interfaces to the operating system or your own applications (e.g. file processing or your own applications for which no host adapter exists).

For more detailed information, refer to the WebTransactions manual “Template Language”.

Using LDAP directory services

The `WT_LdapConnection` class is defined in WTScript in order to provide support for the Internet protocol for directory services LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol). The methods in this class enable you to use LDAP directory services such as:

- Retrieve entries using user-specific search criteria
- Add entries
- Delete entries
- Modify entries
- Compare entries

In addition, the `WT_LdapConnection` class contains methods for setting up and terminating an LDAP session.

For more detailed information, refer to the WebTransactions manual “Template Language”.

Accessing dynamic web contents

Via an HTTP host adapter, you can access the content of any resources that are present in the WWW from within a template. You can then analyze HTML and XML resources with WTScript.

For more detailed information, see the WebTransactions manual “Access to Dynamic Web Contents”.

Web frontend for web services

WebTransactions implements an interface for accessing general web services.

For further information, see the WebTransactions manual “Web Frontend for Web Services”.

Accessing WebTransactions functionality from any client

When you work with WebTransactions, you usually convert host applications into dialog applications for the WWW that will run in any browser. The client interface `WT_REMOTE` makes it possible to access the resources of a WebTransactions application from any client. In the case of Java clients, this functionality is made available in a separate class library.

For further information, see the WebTransactions manual
“Client APIs for WebTransactions”.

2.3 WebTransactions components

The diagram below provides an overview of the architecture of WebTransactions. The integrated components of the WebTransactions supply units are indicated by a gray background.

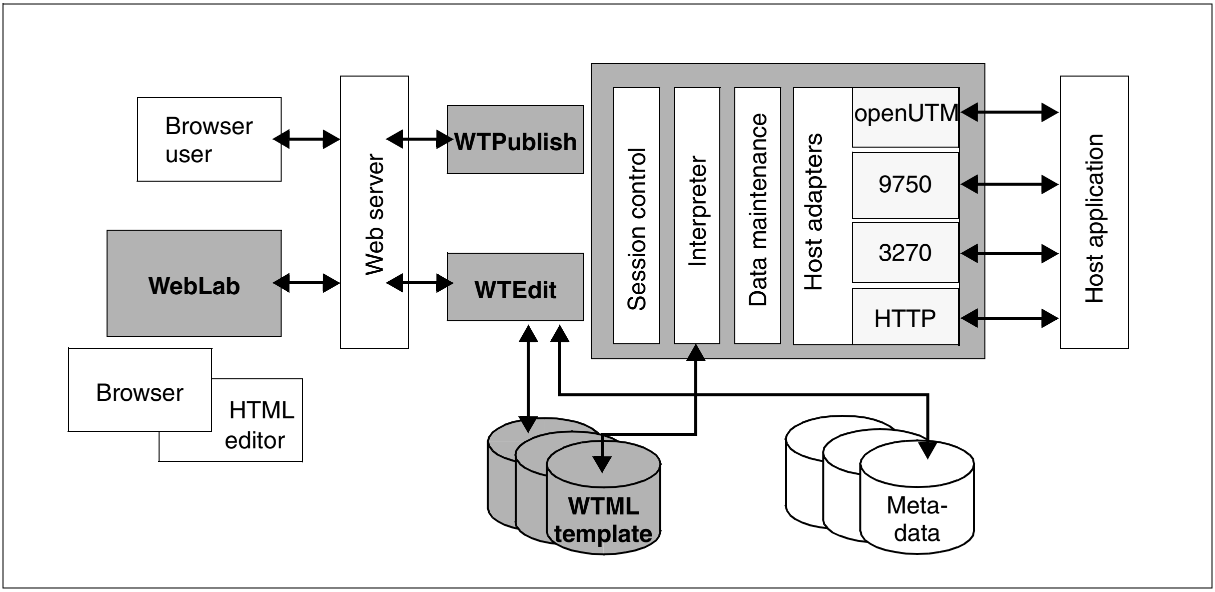


Figure 1: Components of WebTransactions

The transfer of pages and data in the Web takes place on the basis of the HTTP protocol (Hypertext Transfer Protocol) or the HTTPS protocol (Hypertext Transfer Protocol Secure). These standardised protocols are used for accessing WebTransactions, irrespective of whether these are end-user or administration accesses, for which web browsers are used, or whether the access is made via the WebTransactions development system, WebLab.

Using the Web server, WebTransactions communicates via the Common Gateway Interface (CGI), a standard interface for calling programs on a Web server, or via the Microsoft Internet Server Application Program Interface (ISAPI).

WTPublish, WTEdit

The communication interface to the Web server is formed from the following WebTransactions components:

- `WTPublish.exe/WTPublishISAPI.dll` for “normal” access by end users
- `WTEdit.exe` for edit access

This differentiation allows you to implement individual security measures.

Kernel components

The WebTransactions kernel is formed from the following components, which are identical for all supply units:

- session control
- interpreter for the template language WTML (WebTransactions Markup Language and WTScript)
- database for dynamic user data

Host adapters

Special host adapters (communication modules) for communicating with the host applications are provided with the various supply units. These ensure that the host application interfaces can be used without modification. Using this host adapter technique, the specifications of different protocols become transparent for the other WebTransactions components.

The host adapter for dynamic Web content is supplied with each supply unit, this allows you to combine individual host applications with dynamic Web content.

Modular, open architecture - new host adapters

Due to the modular architecture of WebTransactions and its compliance with open standards, new host adapters (communication modules) for additional host application types can be implemented quickly and cost-effectively when required - even within project solutions without this affecting the other components.

Program IFG2FLD

With the supply units WebTransactions for OSD and WebTransactions for openUTM the program `IFG2FLD` is also supplied. This tool is used to prepare for the conversion of FHS formats. This tool creates a description file which is then used as input for WebLab when generating the templates.

3 What is a WebTransactions application

A WebTransactions application provides the link between a host application and the Web. It ensures that data received and transferred by the host application is in a form which can be processed and displayed by a Web browser. The host applications may be different and can be located on different host platforms.

A WebTransactions application provides the end user with a dialog interface in a browser (ultra thin client). For each of the clients connected, WebTransactions manages a session context on the server, which means Web sessions and dialogs are possible with no additional programming. The data source and receiver of data may be one or more autonomous host applications connected to the WebTransactions application.

The communication between WebTransactions and the host application is carried out by a so-called host adapter which is served by the protocol of the relevant host application. The WebTransactions supply unit is named after the protocol used for communication between the host adapter and the host application.

The computer on which WebTransactions is running is also known as the WebTransactions server or integration server. The host application may run on the same server as WebTransactions although this is not usually the case. All stages of the integration of your host application may be carried out locally using WebLab or remotely on the WebTransactions server.

3.1 Components of a WebTransactions application

Alongside the protocol-specific supply unit of WebTransactions, a WebTransactions application requires a base directory. All your application’s files are stored in this base directory. These include:

- start template
- Automask template or format-specific WTML templates used to control conversion between host application and browser
- protocol-specific configuration files

The base directory must always be located on that computer on which WebTransactions is running. You can create the base directory and the required templates using the WebLab development environment, as described in chapter “The WebLab development environment” on page 155.

You can store all the files that have to be accessible directly from the web server when your template is displayed in the base directory. For this reason, the subdirectory `wwwdocs` is created when the base directory is generated. Here you can store images, client-side scripts etc. that have to be accessible directly from the browser. See also section “The wwwdocs subdirectory” on page 60.

The structure of the base directory is dependent on the WebTransactions supply unit being used and is described in section “Structure of the base directory” on page 55.

In order to connect a host application to the WWW, you will, alongside the appropriate WebTransactions supply unit, also require the following:

- a computer on which a Web server is running, this is to be used as the integration server
- as many other computers as you wish on which a browser is running, these are to be used to access the WebTransactions application

A WebTransactions application can, therefore, run on one or more computers using a variety of operating systems: for example, the browser as the client on a Windows computer, the WebTransactions server running on a Unix computer and the connected host application on an OSD or MVS mainframe.

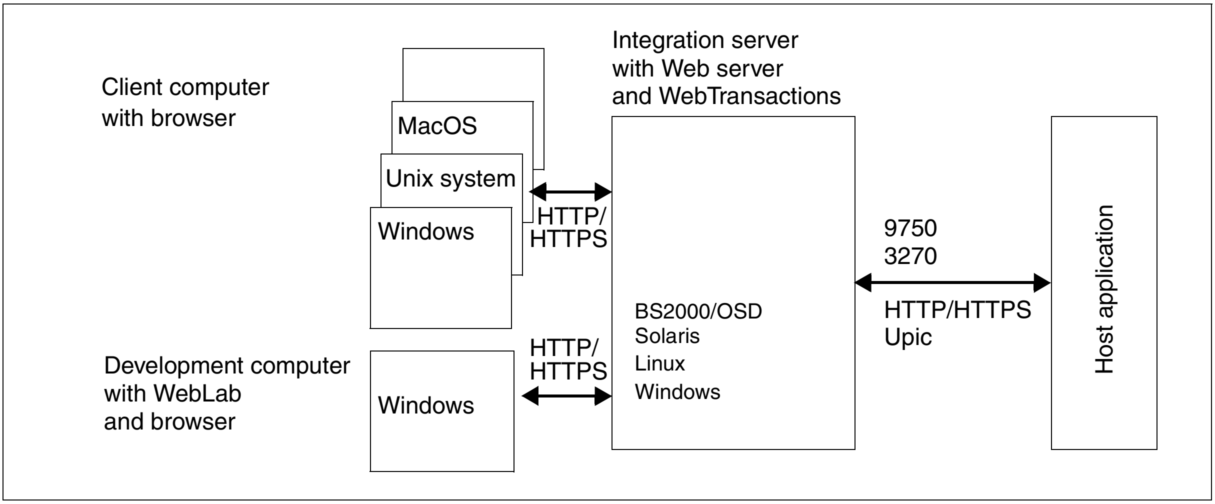

Client computer with browser
MacOS
Unix system
Windows
HTTP/ HTTPS
Integration server with Web server and WebTransactions
BS2000/OSD
Solaris
Linux
Windows
9750
3270
HTTP/HTTPS
Upic
Host application
Development computer with WebLab and browser
Windows
HTTP/ HTTPS

Figure 2: Distribution options for a WebTransactions application

3.2 WebTransactions session

The time period during which a user works with the WebTransactions application is known as a WebTransactions session. At the beginning of the session, WebTransactions starts a process in which the WTHolder program runs. This process continues to exist throughout the entire session. The correlation between the WTHolder program and the user's browser is managed by the CGI program WTPublish.The number of active WTHolder processes thus corresponds to the number of users simultaneously active.

During a WebTransactions session, it is possible to open and close one or more connections to host applications. If several host connections are used in a WebTransactions session, this can take place in succession or in parallel.

For information on starting and ending a WebTransactions session, see the sections "Start options" on page 93 and "Terminating a session" on page 116.

3.2.1 Roaming Sessions

A roaming session is a WebTransactions session, which can be called by different client devices either in succession or simultaneously. For example, a user can continue a session which was started at a workstation, from a mobile device. This mechanism guarantees a continuous operation even after a client device has crashed. A roaming session thus supports a restart from the server.

With WebTransactions, the user can determine at the start of a session if the session may also be called from other client devices. To do this, the user needs a personal session ID.

Roaming sessions can also be used in a cluster (see section "Cluster concept" on page 142).

WTBean wtcRoaming

The WTBean `wtcRoaming` is supplied to support the roaming sessions. This WTBean generates a start template for a roaming session and saves user authentication data at the beginning of a session. If the connection to a roaming session is re-established, the WTBean expects the same authentication data as at the beginning of the session.

WTBean `wtcRoaming` is described in the WebLab Online Help. It describes how to create a start template for a roaming session with `wtcRoaming` and how to test the restart.

WTBean wtcSingleSignOn

The WTBean `wtcSingleSignOn` provides the Single Sign-On function. You must login using only the User ID and password at WebTransactions and start the subsequent processes using the corresponding authentication.

The WTBean `wtcSingleSignOn` provides a very convenient support for roaming sessions due to its user concept and identification check.

➪ You can obtain the WTBean `wtcSingleSignOn` from the download area of the WebTransactions home page under the keyword “Ready-to-run”. WTBeans must be installed separately on the host on which WebLab runs.

i We recommend that you use the WTBeans `wtcRoaming` or `wtcSingleSignOn` to create start templates and to check the authentication data. The following text describes what you have to be aware of if you decide not to use WTBeans.

Starting a roaming session

To start a roaming session, which can be accessed by several client devices, the value of the attribute `WT_SYSTEM_SESSION` containing the session's ID, must start with the prefix `R` (to start a session see section “Starting a WebTransactions dialog application” on page 92).

Example

```
http://myhost/cgi-bin/WTPublish.exe/startup?
WT_SYSTEM_BASEDIR=C:/Base/App&WT_SYSTEM_FORMAT=start&
WT_SYSTEM_SESSION=R-MyRoamingId
```

In this example, the session ID for the roaming session is `R-MyRoamingId`.

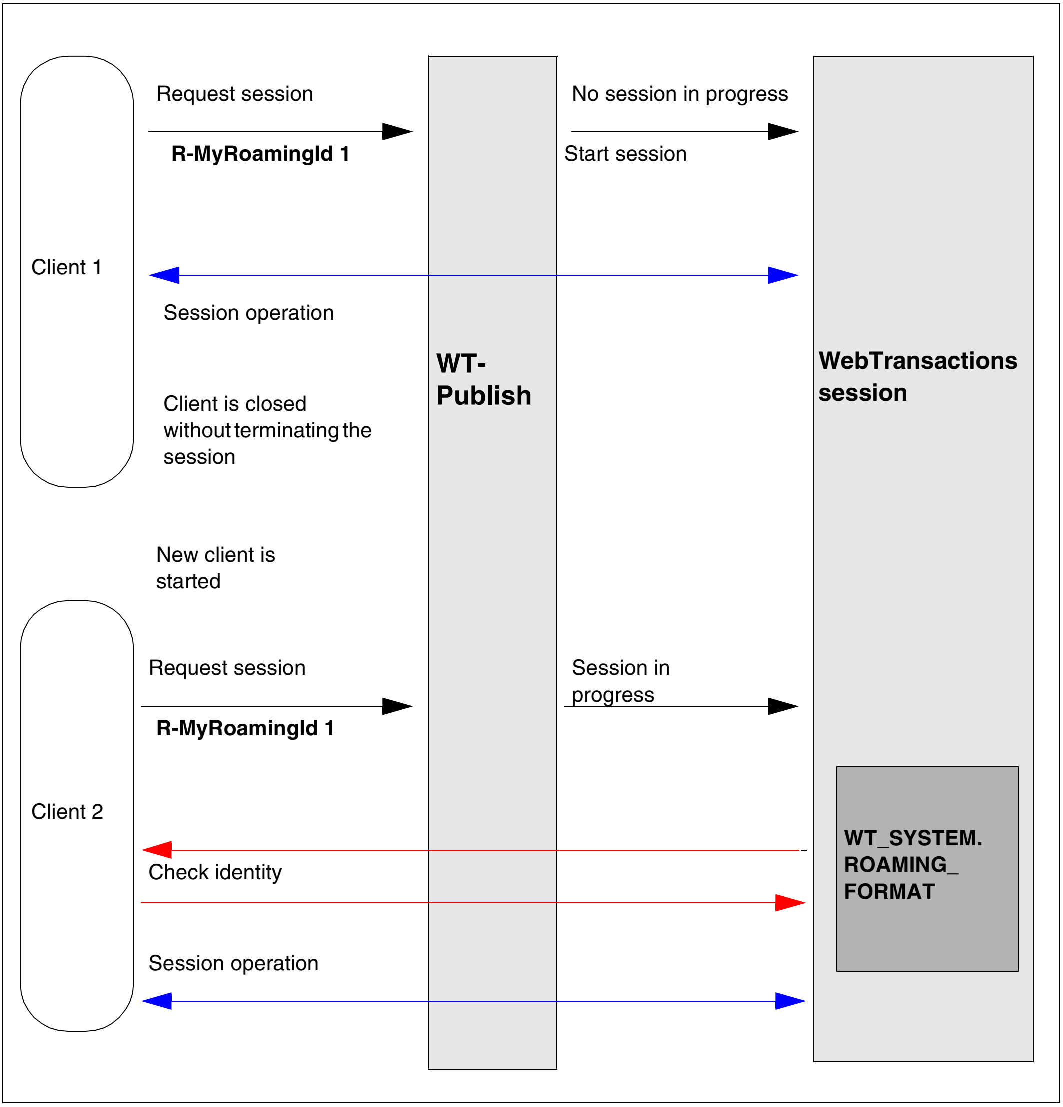
The session is started using the start template indicated.

If a session with the indicated session ID is already in progress, it is continued at the last synchronously generated page.

To prevent unauthorised access to the session, the user's authenticity must be verified. The attribute `WT_SYSTEM.ROAMING_FORMAT` is used, which has to be provided with a template name. The authorized access to the session must be controlled with this template. For instructions on implementing such a template, see section “Checking identity” on page 44.

i On Windows platforms, each session ID of a roaming session can only be used once. In Windows, two sessions with the same ID cannot be operated, even if the sessions run in different base directories.

The following figure shows the procedure for starting a roaming session:



Checking identity

When continuing a session already in progress, the user's authenticity must be re-checked, otherwise unauthorised access to the session is a possibility. For this purpose you can either evaluate the attached request data or send an authentication page.

When re-entering a session, WebTransactions sets the attribute `WT_SYSTEM.ROAMING` to `true`, so the status can be reliably verified in other templates as well. Using one template, it can be distinguished between a new login or the continuation of a session. Thus, it is possible to use the start template for repeated authentication and to omit different initializations by using the value `WT_SYSTEM.ROAMING` when a session is continued.

When access is denied, you should note the following:

- The attribute `WT_SYSTEM.SIGNATURE` must not be transmitted to the browser. Otherwise, unauthorised access to the session is possible.
- When calling another form, you should use the `wtDataform` tag to generate the respective URL automatically.
- Do not use the attributes `HREF` and `HREF_ASYNC`.

In case of a positive verification of identity, normally the last synchronously generated page of the WebTransactions application must be displayed.

Example

```
<script>
  document.location = '##WT_SYSTEM.HREF_ASYNC#';
</script>
```

The output of the last synchronously generated page does not make sense if, for example, a different device is used on the client side. In this case, the page must be generated from scratch. In this case, the execution of the `wtOnReceive` scripts may sometimes be omitted. The template for the new verification of the authenticity could, for example, switch the style according to the currently used client device. A new generation using the modified style is triggered by outputting the following script.

Example

Attaching `WT_DISPOSE_RECEIVE_SCRIPTS=true` in the following script suppresses the execution of all `wtOnReceive` scripts.

```
<script>
  document.location = '##WT_SYSTEM.HREF#&WT_DISPOSE_RECEIVE_SCRIPTS=true';
</script>
```

Roaming sessions in WebLab

Every session, which was started with WebLab (see section “Starting a session” on page 161), is also a roaming session.

WebLab provides the option of testing the restart of a roaming session. Use the command **Control/Test Roaming**. This command triggers the **Test Roaming** dialog box. In this dialog box, additional start parameters can be modified, for example in order to test false login information.

The **Test Roaming** dialog box is described in the WebLab Online Help.

Roaming sessions in a cluster

Roaming sessions can also be used in a WebTransactions cluster (see section “Cluster concept” on page 142).

To identify a roaming session within a cluster, the session ID must also be transmitted at the start of a cluster session (see section “Starting a cluster session” on page 146).

Example

```
http://my-server/cgi-bin/WTCluster.exe/my-cluster-id?
WT_SYSTEM.SESSION=R-MyRoamingId
```

3.2.2 Service applications

A service application is a WebTransactions session, which can be called in turns by different users. Service applications are mainly used for dialogs, which are comprised of one step only. They are used, for example, for frequent retrieval actions involving large amounts of data.

Such sessions are exclusively operated in a non-synchronised dialog (see section “Non-synchronized dialog” on page 64). Service applications start automatically if required.

Starting a service application

A service application is called by entering the URL in the browser or by entering the URL in a form (see section “Starting a WebTransactions dialog application” on page 92):

```
http[s]://machine/cgiPath/WTPublish.exe/startup?
WT_SYSTEM_BASEDIR=basedir&WT_SERVICE_PAGE=service-template&par1=val1&...
```

machine
Internet address or symbolic name of the computer, where WebTransactions is installed (with port number for the HTTP server if required).

cgiPath
Path (prefix) for CGI programs set with the HTTP server

basedir
Base directory, where the WebTransactions service application is installed. *basedir* is an absolute path (Windows with drive identifier)

service template
Template containing the service. The name of the template must end with the suffix `.service`.

Example

```
basedir/config/forms/myService.service
```

par1=val1
Under other name/value pairs you can set more parameters for the service application.

When `WTPublish` is called by a service application, it first tries to locate a vacant WebTransactions session, which was started for providing the services. Session IDs of service applications receive the prefix `WTSVC` and a continuous index.

Example

```
WTSVC-c__basedir_mybasedir-1
WTSVC-c__basedir_mybasedir-2
...
```

If all sessions in progress are busy, a new session is started for the service. This session processes the request and is available again after its completion.

i The attribute `WT_SYSTEM.TIMEOUT_USER` remains in effect for sessions with service applications. Additional sessions started during periods of high load thus can be terminated automatically during periods of low load.

Templates for service applications

Templates implementing a service application and referenced as `WT_SERVICE_PAGE` must end with the suffix `.service`.

Since the dialog in a service application has one step only, no information can be saved from one request to the next. Therefore, the following must be considered when programming such templates:

- The attributes `WT_SYSTEM.HREF_ASYNC` and `WT_SYSTEM.HREF` are not supplied with `SESSION`, `SIGNATURE` and `FORMAT_STATE`.
- `WTDataform` is always generated for the non-synchronised dialog.

Service applications in WebLab

To start a service application in WebLab, use the standard command **File/Start session** (see section “Starting a session” on page 161). In the **Start session** dialog box, enter the template for the service application as the start template.

3.3 Templates

A template is a model for the generation of specific code. It contains fixed parts that are taken over on generation and variable parts that are replaced by the corresponding current values.

This section describes the following template types:

- WTML templates, the associated language resources, the definition of the operating steps within the template, and the template during execution in the dialog cycle
- Master and class templates for the cross-application, uniform implementation of certain format areas or objects
- Module templates for the global definition of classes, functions and constants for an entire session
- WTBeans as re-usable components for template programming

3.3.1 WTML templates

WTML templates (or simply templates for short) form the core of each WebTransactions application. At runtime, they are evaluated by WebTransactions and define the layout of the HTML pages displayed in the browser, control the WebTransactions application, and define the steps for communicating with the host applications.

The Automask templates (OSD, MVS) or format-specific templates (OSD, MVS, openUTM) use default specifications to create a HTML page from each unit of the host user interface (alphanumeric format). These automatically generated templates can be used unchanged, or as the basis for further customization. The template language WTML (WebTransactions Markup Language) is provided for programming the templates. The WebLab development environment also offers a convenient means of editing and testing templates.

By default, each format-specific template is stored in a separate file *formatname*`.htm` in the directory *basedir*`/config/forms`. However, you can create further template directories for different layouts and languages, see section “Subdirectories for style and language variants” on page 56.

WTML template language resources

The following language resources can be used in the templates:

- Standard HTML tags, text, and client-side JavaScript and all other language resources that can be interpreted by the browser

  In the templates, you can use any HTML tags and `<SCRIPT>` tags or constant text which can be interpreted by the user’s chosen browser. All JavaScript language resources that can be interpreted by the user’s chosen browser are thus supported.

  Like the standard HTML tags, these scripts are interpreted by the browser rather than by WebTransactions, and are thus also known as client-side JavaScripts. You can also use JScript or VBScript in Microsoft Internet Explorer. These HTML areas of the template (HTML tags and text) are sent to the browser unchanged. From the viewpoint of WebTransactions, they are part of the HTML area.

- WTML tags

  WTML tags allow you to dynamically generate HTML pages and control the host application. They are used to calculate values, for instance, or to transmit information from the host application.

  Areas containing WTML tags are interpreted by WebTransactions and the result is sent to the browser. They do not therefore form part of an HTML area.

- WTScripts

  Like client-side JavaScripts, WTScripts are contained in areas delimited by special tags. Instead of HTML `SCRIPT` tags, however, you must use the WTML tags `wtOnCreateScript` and `wtOnReceiveScript`. This indicates that these scripts are to be executed by WebTransactions as opposed to the browser, and allows you to signal the desired execution time. Similar to OnCreate actions, OnCreate scripts are executed before the page is sent to the browser. OnReceive scripts, on the other hand, are not executed until a response is received from the browser.

  Like WTML tag areas, WTScript areas are interpreted by WebTransactions and the result is sent to the browser. WTScript areas do not therefore form part of the HTML area.

- Evaluation operators

  Evaluation operators can be used to query the current values of objects or object attributes, see section “Objects - dynamic data” on page 66. You can write any expression as an evaluation operator. This is then evaluated and the result output.

The WTML template language resources are described in detail in the WebTransactions manual “Template Language“.

OnCreate and OnReceive times

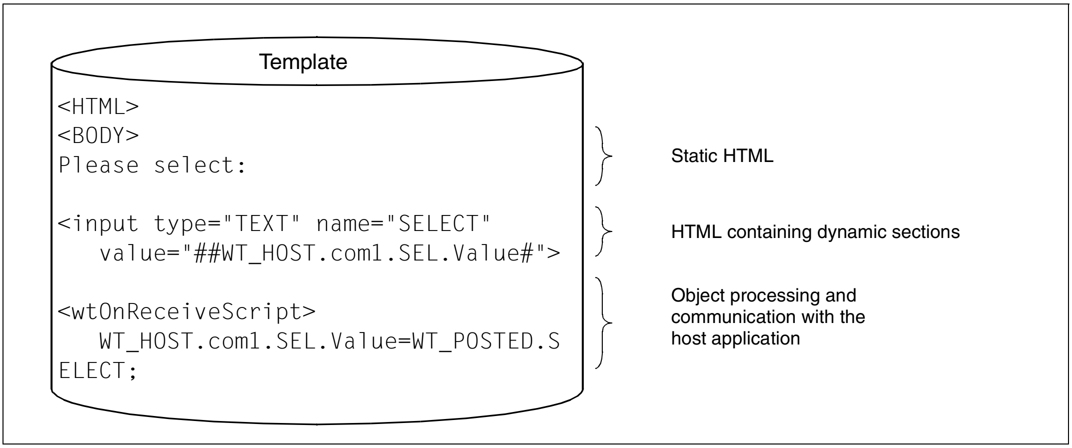
The template not only defines the processing steps to be performed immediately by WebTransactions when generating the HTML page (i.e. OnCreate), but also specifies the processing steps to be performed when the data posted by the browser is received (OnReceive) and in response to user input. These OnReceive processing steps are initially buffered by WebTransactions, and are not executed until after the current HTML page is generated, sent to the browser, and the data posted by the browser is received.

This mechanism allows you to define a complete dialog cycle in a single template (see also section “Dialog cycle” on page 62):

- creation of the HTML page sent to the browser
- post-processing of data posted by the browser in response to this page
- transmission of the post-processed data to the host application, and receipt of the next host message

Although the template is only interpreted once by WebTransactions, the different processing steps come into effect at different times.

Sample template



```
<HTML>
<BODY>
Please select:

<input type="TEXT" name="SELECT"
   value="##WT_HOST.com1.SEL.Value#">

<wtOnReceiveScript>
   WT_HOST.com1.SEL.Value=WT_POSTED.S
ELECT;
```

Template
Static HTML
HTML containing dynamic sections
Object processing and communication with the host application

This template contains both the HTML definitions for output on the browser, and the statements for processing objects and communicating with the host application.

The HTML page sent to the browser appears as follows:

- The text `Please select:` appears as a static area.

  The input field is created. It is called `SELECT` and is preset to the value contained in the `Value` attribute of the host data object `SEL`.

- The WTML tag `<wtOnReceiveScript>` starts a WTScript area containing the processing steps to be executed at OnReceive, i.e. after the browser has posted the user input.

  Through a simple assignment, the value stored in the `SELECT` attribute of the posted object `WT_POSTED` is transferred to the `Value` attribute of the host data object `SEL`. The value entered by the user is now contained in the host data object.

  A `send` call is issued which transfers the current host data object `SEL` to the host application together with all data belonging to this message. The next host message is then retrieved by means of a `receive` call. The statement `setNextPage` is used to determine the next template to be processed. This is specified using the system object attribute `FLD`.

3.3.2 Master, class and module templates

Master and class templates provide a consistent application-wide conversion of particular areas or objects in the formats. Using module templates, you can globally define classes, functions and constants for an entire session.

3.3.2.1 Master templates

Master templates are used by WebTransactions when generating the Automask and the format-specific templates and ensure that the layout is consistent.

The master template concept is particularly effective for host applications in which many of the formats have a similar structure: for example, a fixed format for the header, working and footer areas. In a situation like this, it is enough to specify the structure in a master template and to then assign this master template when generating both the format-specific templates and the Automask. All generated templates automatically have the desired structure.

Master templates, like any other template, can contain fixed HTML areas as well as any WTML tag and WTScripts. Master templates also have special master template tags, MT tags for short. These are described in the WebTransactions manual “Template Language“.

Separate master templates are supplied with each of the WebTransactions supply units which you can modify to suit your requirements or just use unchanged. You can specify which master template is to be used as the master template during generation.

3.3.2.2 Class templates

Class templates can be used to automate the evaluation of host data objects. Instead of writing the same statements, over and over again, for each of the host data objects, you can define class templates.

A class template in WebTransactions contains valid, recurrent statements for the entire object class (e.g.input or output field). Class templates are worked through if the evaluation operator or the toString method is applied to a host data object.

Class templates are inserted in their entirety into the calling template. You can use the same language resources in class templates as you can for all other templates.

Class templates have the file name suffix `.clt` (class template) and are, by default, stored in the directory *basedir*`/config/forms` just like normal templates. You can also implement various styles and languages for class templates, whereby the same search strategy is used as for normal templates, see also section “Search strategy” on page 57.

For more information on class templates, see the WebTransactions manual “Template Language“.

3.3.2.3 Module templates

In module templates, you can define classes, functions and constants for WTScript which you can then access in WTScripts and evaluation operators throughout the entire WebTransactions session. In this way, you can, for example, import WTScript functions and class libraries. You should note that classes, functions and constants must be defined in `wtOnCreate` scripts.

A module template is loaded using the `import()` function. WebTransactions searches the corresponding template according to the specified language and style, (see section “Search strategy” on page 57).

3.3.3 WTBeans

WebTransactions provides you with reusable components, known as WTBeans, for template programming. A distinction is made between inline and standalone WTBeans:

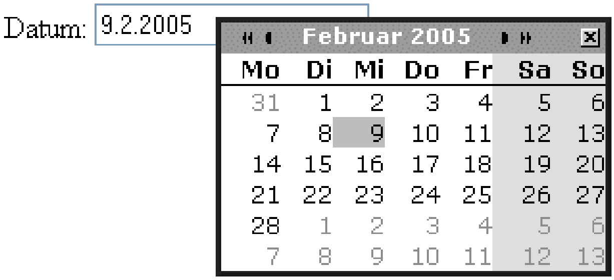
- Inline WTBeans correspond to a part of a WTML document
- Standalone WTBeans represent an autonomous WTML document

WTBeans can also be differentiated between on the basis of their function:

- WTBeans for browser dialogs
- The components control the framework within which one or more dialog steps are performed. For example, they define whether the dialog is conducted in a single window, multiple windows or in different frames.
- WTBeans for browser dialog elements
  The components are primarily used to display data in the browser, e.g. for GUIs which can be defined by multiple HTML tags and by client-side Javascript.

  Example

  The WTBean `wtcPopupDate` provides a calendar window in which the date can be selected quickly and conveniently with the mouse.



Datum: 9.2.2005

Mo	Di	Mi	Do	Fr	Sa	So
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	1	2	3	4	5	6
7	8	9	10	11	12	13

- WTBeans for processing
  The components do not generate a GUI but instead contain an item of processing logic, for example for communications with a host or interfacing with a user-exit library.

With WTBeans, WebTransactions provides you with a set of reusable components with which you can control communications both with the browser and with the host application. In WebTransactions as shipped, only the WTBeans for communication with the host application are installed. For more information, refer to the descriptions in the protocol-specific manuals.

You can download additional WTBeans from the download area of the WebTransactions home page under the keyword “Ready-to-run”. WTBeans must be installed separately on the computer on which WebLab is running. For more information, refer to the relevant documentation.

Components of WTBeans

Every WTBean has a unique name. As this name is used as a variable name, it is necessary to differentiate between uppercase and lowercase. The names begin with `wtc` (for **WebTransactions C**omponent).

Example

```
wtcStartOSD
```

Every WTBean is defined in a description file which corresponds to a template. The name of the description file consists of the name of the WTBean and the suffix `.wtc`. Alongside the description file, there may be other files associated with a WTBean such as image files, files containing exported scripts etc.

Example

```
wtcStartOSD.wtc
```

By means of its properties, every WTBean can be adapted to the WebTransactions application in which it is used. To this end, the properties are used to generate a GUI for WebLab in which you can edit the properties simply and conveniently.

A WTBean’s description file contains:

- the prototype of an HTML page, a template or a template wizard
- a description of the used resources
- a description of the GUI for parameter editing
- a list of help files and target paths in the base directory

Following installation, the supplied WTBeans are located in the `wtcCollection` subdirectory of the WebLab installation directory where they are stored in the corresponding subdirectories `inline` or `standalone`. These directories themselves contain subdirectories with the names of the corresponding WTBeans.

When you use your first WTBean in WebLab, the `wtcUsage` subdirectory is created in the base directory. All the WTBeans that you use for template processing are copied to `wtcUsage`. This means that you are also able to edit these WebLab components when working at another computer.

3.4 Structure of the base directory

A base directory (*basedir* in the various syntax specifications) contains all the files required to link an application to the Web.

Information on the structure of base directories that apples to only one individual supply unit can be found in the corresponding User Guide.

WTHolder (Windows / Unix platforms)

The base directory contains a link with the name `WTHolder` which points to the corresponding program in the installation directory:

Windows platform	*install_dir*`/lib/WTHolder.exe`
Unix platform	*install_dir*`/lib/WTHolder`

The WTHolder program is started in a separate process for each user of a WebTransactions application. It remains active throughout the entire session, and controls the parsing of templates and communication between the browser, WebTransactions, and the host application. It also ensures that modules are loaded from the shared libraries if required.

Optional: shared libraries (Windows / Unix platform)

WebTransactions makes the individual host adapters available as shared libraries (e.g. `WTCommOSD.dll` under Windows or on the Unix platform `WTCommOSD.so`). These libraries are not present under OSD as this functionality is integrated in `WTHolder` here.
These libraries also contain the supplied user exits.

These shared libraries are **not** automatically created in the base directory when the base directory is generated. By default, WebTransactions uses the libraries in the installation directory at runtime. Copies of these libraries in the base directory are only of any use if you need to use versions different from those in the installation directory. This is the case, for example, if you have extended the supplied user exit library with your own user exits or have generated completely new user exit libraries.

Statically linked WTHolder programs (BS2000 / OSD)

Under POSIX it is not possible to use shared libraries. For this reason, the host adapters are statically linked to the WTHolder programs
(`WTHolderUTM`, `WTHolderUTMV4`, `WTHolder...`).

For this reason, user exits must also be statically linked (see the WebTransactions manual “Template Language“).

Subdirectories

The base directory also contains the subdirectories `msg`, `tmp`, `wtcUsage` and `wwwdocs` which are described in the following sections.

You can also create additional subdirectories yourself, for example in order to archive master templates or static HTML pages.

3.4.1 The config subdirectory

The `config` directory contains templates in the `forms` subdirectory which are used to create the user interface.

3.4.1.1 The forms subdirectory

The generated templates are stored by default in the `config/forms` subdirectory. This subdirectory is always present and should contain a template for each format adapted individually. The templates stored in `forms` represent the default style and default language for the customized templates.

The `forms` subdirectory also contains the predefined templates which are provided by WebTransactions.

3.4.1.2 Subdirectories for style and language variants

It is possible to implement different style and language variants of the same logical interface of a host application in order to address the needs of individual user groups. These template variants are stored in separate directories.

The directories then also contain the corresponding start templates. The start template is the first template that is read when a WebTransactions session is started (see section “Starting a WebTransactions dialog application” on page 92). To enable WebTransactions to find a start template in another style or language, you must set the system object’s `STYLE` and `LANGUAGE` attributes to the relevant value at session start. In this way, you can control the layout used by WebTransactions.

Different style variants (WT_SYSTEM.STYLE)

The following conditions apply for different style variants:

- The templates for the default style must be stored in the `config/forms` subdirectory.
- The templates for the other interface styles must be stored in a subdirectory created under `config` (parallel to `forms`). The name of this subdirectory is freely selectable. This name is entered in the `STYLE` attribute of the system object.

  You can create any number of directories containing templates in another style. For instance, you may wish to develop variants containing more or fewer graphical elements for different browser configurations.

Different languages (WT_SYSTEM.LANGUAGE)

If you wish to offer interfaces in the same style but in different languages, you must create additional directories for the language variants under the style directories. The name of a language directory is freely selectable, but must be specified in the LANGUAGE attribute of the system object. This allows you to define the language you wish to use.

Search strategy

You do not need to offer all your templates in all styles or languages variations. WebTransactions uses a search strategy in order to locate the appropriate template.

The selection of a template is controlled by means of the system object attributes `BASEDIR`, `STYLE`, `LANGUAGE`, `FORMAT`, and `DEFAULT_FORMAT`. WebTransactions searches for the appropriate template in accordance with the following strategy:

BASEDIR-value`/config/{`*STYLE-value*`|forms}[/`*LANGUAGE-value*`]/`*FORMAT-value*`[.htm]`

- `BASEDIR` specifies the base directory.
- The name part `config` is always automatically appended to the base directory.
- This is followed by the path section taken from the system object attribute `STYLE`. If no value is specified in `STYLE`, the default value `forms` applies for this path section.
- This is followed by the path section taken from the system object attribute `LANGUAGE`. If no value is specified in `LANGUAGE`, this path section is ignored.
- The name of the template file is contained in the system object attribute `FORMAT`. The suffix `.htm` may be omitted.

WebTransactions begins by searching for the template in the following directory:

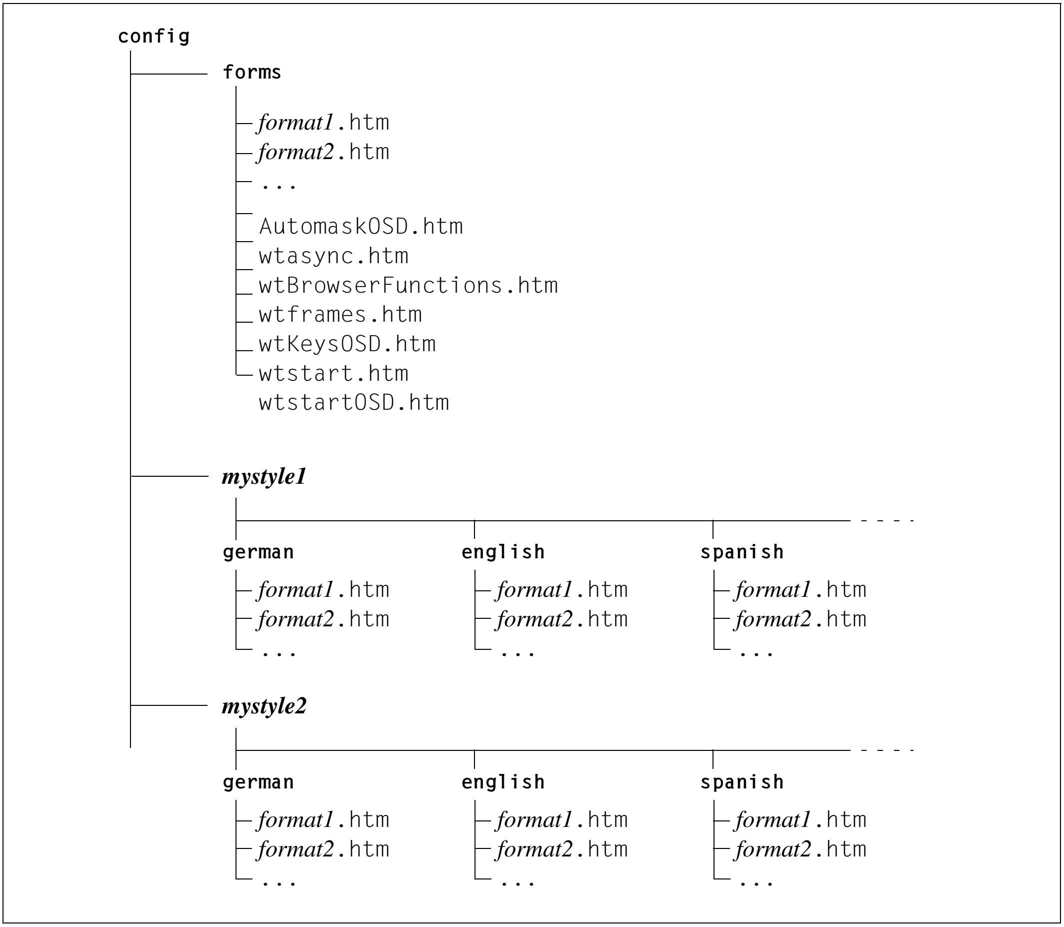
1. *BASEDIR-value*/config/*STYLE-value*/*LANGUAGE-value*

As it is not necessary to output each variant of each template, WebTransactions searches the directories listed below in the specified sequence if it does not find the template in the above directory.

2. *BASEDIR-value*/config/*STYLE-value*
3. *BASEDIR-value*/config/forms/*LANGUAGE-value*
4. *BASEDIR-value*/config/forms

If the template specified in FORMAT is not found in any of these directories, WebTransactions attempts to load the template specified in `DEFAULT_FORMAT`. The same search strategy is used here.

Example: Style and language variants



3.4.2 The msg subdirectory

The `msg` subdirectory contains links to the message files in the installation directory (`errmsgs`, `errmsgs*`) and to the files used to display messages in the browser, the error message templates `loctmpl` and `errtmpl`:

- WebTransactions uses `loctmpl` to display errors whose cause can be located at a particular position in a template. The name of the template in question, the line number, and the column are specified.
- `errtmpl` is used to display all the errors whose cause cannot be located in a template.

All the files exist for both English messages (no suffix) and German messages (with the suffix `.de`).

If you modify the messages for a WebTransactions application with WebLab then a copy containing the changes is stored in *basedir*`/msg` when you save. The reference to the corresponding file in the installation directory is deleted.

Otherwise, the modified messages would apply to all the WebTransactions applications on the server. This is the case for both `loctmpl` and `errtmpl`.

WebTransactions allows you to make the messages available in different languages. You can set the language via the system object's `LANGUAGE` attribute. WebTransactions then uses the appropriate error message file `errmsg.`*lang* (or `errmsg*.`*lang*) together with the error message templates `loctmpl.`*lang* and `errtmpl.`*lang*. Here, *lang* corresponds to the current value of the system object's `LANGUAGE` attribute.

3.4.3 The tmp subdirectory

This subdirectory is used to store temporary files. A *session*`.info` file containing information about the session is created for each WTHolder task. This information can then be evaluated using the administration functions.

In addition, a *session* subdirectory is created for each link in `tmp`. This contains any trace files or temporary files which are created at runtime. It may, for example, contain HTML files for frame support or dynamically generated graphics which are to be integrated in the HTML page.

The name *session* always corresponds to the value of the system object's `SESSION` attribute.

3.4.4 The wtcUsage subdirectory

This subdirectory is not created when the base directory is generated. Instead, it is not created until you use WTBeans. The description files of the employed WTBeans are stored in `wtcUsage`.

3.4.5 The wwwdocs subdirectory

In addition to the templates in the base directory, a WebTransactions application has files which are needed to construct HTML pages and which the Web server must be able to access directly (e.g. JavaScript files or image files). You should place these files in the `wwwdocs` directory which can be accessed by the web server.

If you save all your application's files in the base directory then it is an easy matter to transfer the application from one host to another (see section "Transferring and distributing a WebTransactions application" on page 198).

The `wwwdocs` directory is physically located below the web server's document directory. A symbolic link pointing to `wwwdocs` in the web server's document directory is created in the base directory. This means that all the files which you save in the `wwwdocs` subdirectory of the base directory are physically located in the web server's document directory. In WebLab they are treated as a component of the base directory and can consequently be packed in the archive for transfer and then be unpacked at the target host.

`wwwdocs` and the symbolic link in the base directory are created when the base directory is generated. `wwwdocs` has the following structure which you can extend to meet your own requirements:

```
wwwdocs
 |
 ├applet           Applets
 ├class            Java class files
 ├html             HTML pages, e.g. start pages
 └image            Image files
 └javascript       Client-side Javascripts
 └style            Stylesheet definitions
```

Accessing files in wwwdocs

For the purposes of template programming, you can access the files in the `wwwdocs` directory via the system object attribute `WWWDOCS_VIRTUAL`. `WWWDOCS_VIRTUAL` corresponds to the virtual path under which the base directories of your WebTransactions applications are located. You assign this during administration of the WebTransactions server. For more information, refer to section “Managing the WebTransactions server” on page 139.

Example

```
<img src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/image.gif">
```

You can directly address the files under `wwwdocs` in the web server’s document directory as follows:

document-root/virtual path/basedir/`wwwdocs`/*filename*

document-root Path of the web server’s document directory; you specify this path when installing WebTransactions

virtual path Virtual path which you assign for the directory under which the base directories of your WebTransactions applications are located during administration of the WebTransactions server. For more information, refer to section “Managing the WebTransactions server” on page 139.

basedir Name of the corresponding base directory.

3.5 Dialog cycle

A dialog session can be regarded as a sequence of dialog cycles. Depending on the type of dialog, (synchronized, non-synchronized or remote) a dialog cycle comprises several phases.

For more detailed information on the dialog cycle, refer to section “Dialog between WebTransactions and the browser” on page 110.

3.5.1 Synchronized dialog

Seen from a WebTransactions point of view, a **dialog cycle** consists of the following three phases:

1. Interpreting the template / generating HTML (WebTransactions)

   WebTransactions interprets the template and from this generates the next HTML page to be sent to the browser. The HTML tags contained in the template are transferred unchanged to this HTML page. This procedure is driven by processing steps defined in the template in special WTML tags and WTScripts, which allow dynamic data to be entered in the static HTML sections. These steps serve to convert host data into HTML and to communicate with the host application. An HTML page is thus formed from HTML areas and WTML areas, and is sent to the browser.

   Not all processing steps defined in the template are effective when generating the HTML page (at the time of “OnCreate”): processing steps defined in OnReceive scripts are initially buffered by WebTransactions and are not executed until the third phase of the dialog cycle.

2. Entering data or making a selection (user at the Web browser)

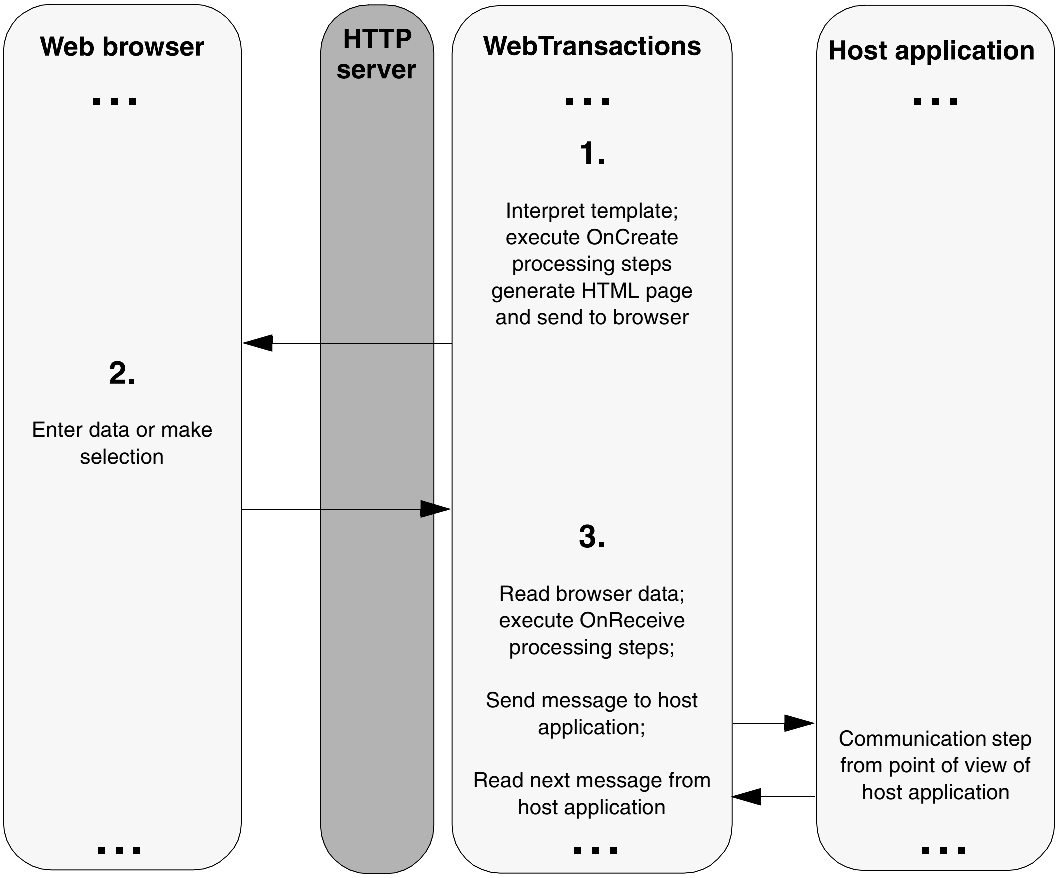
   The user can now make the desired entries in the page displayed in the Web browser. The HTML page is then sent from the Web browser back to the HTTP server, where it is posted to WebTransactions. WebTransactions checks whether the data sent is up-to-date. If an older page is received from the session, then the page generated in the first phase of the dialog is sent to the browser again. If the received data is up-to-date, then phase three is carried out using this data.

3. Executing OnReceive processing steps (WebTransactions)

   WebTransactions now executes the OnReceive processing steps buffered in phase 1. For instance, this may involve mapping the data sent by the browser to the host application format, and then sending this data to the host application. In the last OnReceive processing step, the next message is generally read by the host application. The follow-up template is specified using the `setNextPage()` function which writes the name of the next template that is to be processed in the system object attribute `WT_SYSTEM.FORMAT`.

The default dialog sequence controlled by the host application can also be actively modified (see section “Active dialog” on page 109).

The diagram below illustrates a synchronized dialog cycle:


Web browser
...
2.
Enter data or make selection
...
HTTP server
WebTransactions
...
1.
Interpret template; execute OnCreate processing steps generate HTML page and send to browser
3.
Read browser data; execute OnReceive processing steps;
Send message to host application;
Read next message from host application
...
Host application
...
Communication step from point of view of host application
...

This dialog cycle is part of the synchronized dialog in which WebTransactions expects a specific set of data from the browser in order to process the outstanding OnReceive rules. If the user sends an HTML page which is not the one the system expects to receive, then the most recently generated HTML page is again sent back to the browser.

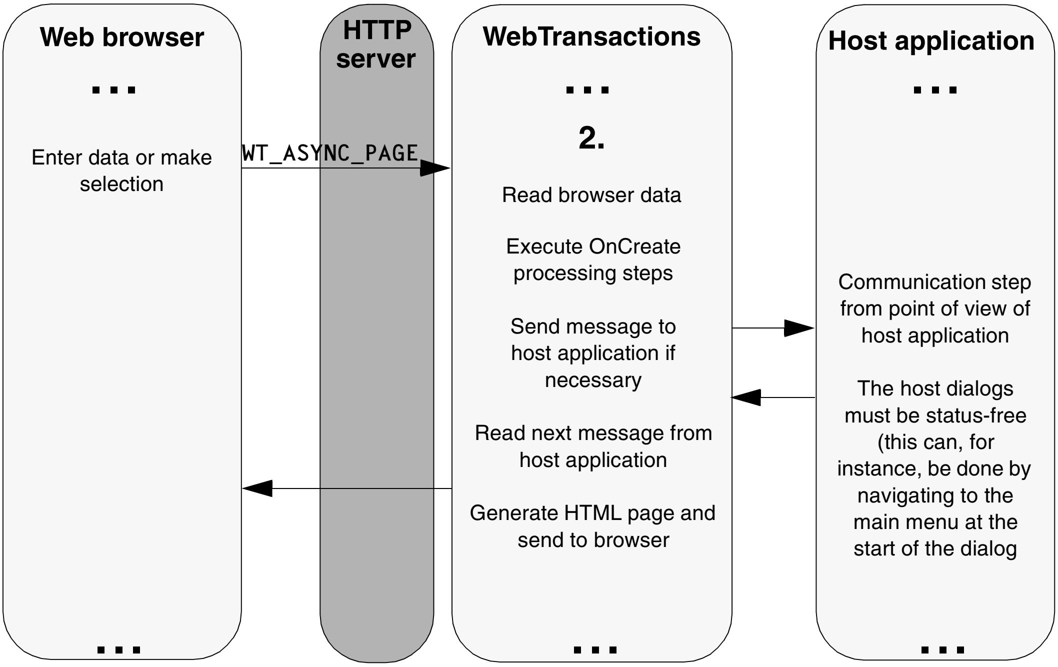
3.5.2 Non-synchronized dialog

If the dialog is non-synchronized, no check is performed to determine whether the data is up-to-date (phase 2 in the synchronized dialog). A non-synchronized dialog consists of possible one-step dialogs without a defined order. It begins with a request from the browser which must specify the template to be executed. The browser may also post other data.

Every non-synchronized template called generates an HTML page, after processing the OnCreate rules. This page is then sent to the browser. This action terminates the non-synchronized dialog and the data sent by the browser is not post-processed using OnReceive rules. This means that data sent by the browser can only be processed using On Create rules.

This procedure means that the dialog cycle for a non-synchronized dialog is reduced to the step “Interpret Template/Generate HTML”. Further information about synchronized and non-synchronized dialogs can be found in section “Dialog between WebTransactions and the browser” on page 110.

The diagram below shows a non-synchronized dialog cycle:


Web browser
...
Enter data or make selection
...
HTTP server
WT_ASYNC_PAGE
WebTransactions
...
2.
Read browser data
Execute OnCreate processing steps
Send message to host application if necessary
Read next message from host application
Generate HTML page and send to browser
...
Host application
...
Communication step from point of view of host application
The host dialogs must be status-free (this can, for instance, be done by navigating to the main menu at the start of the dialog
...

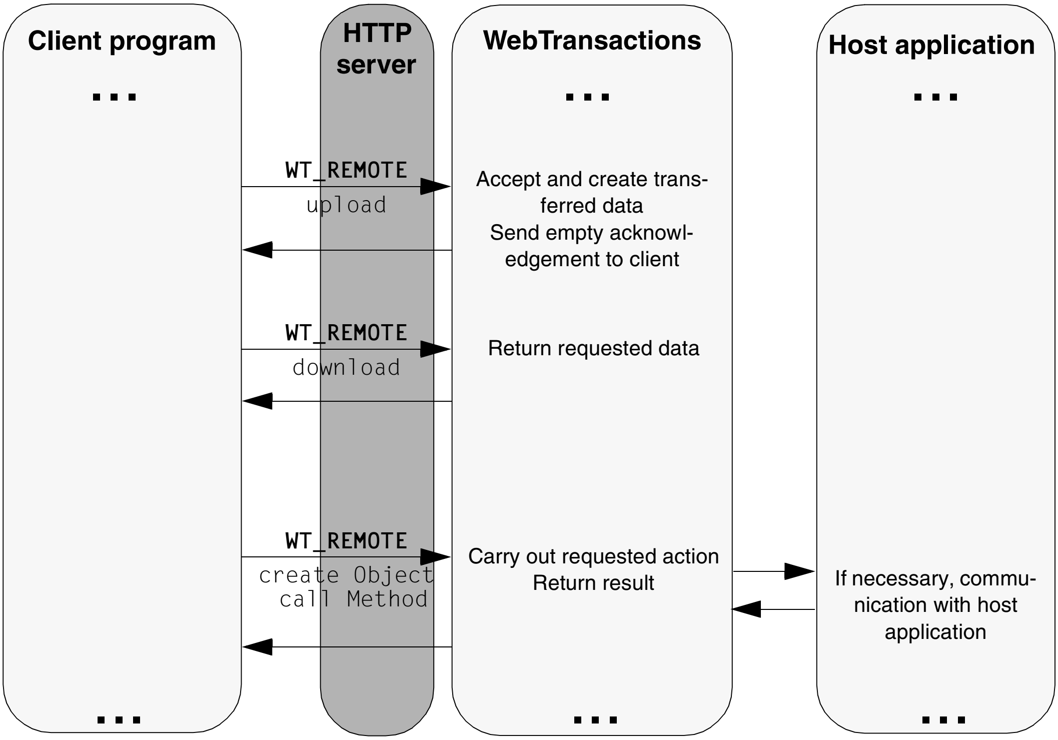
3.5.3 Dialog via client interface

No HTML pages are generated when WebTransactions is accessed via the client interface `WT_REMOTE`. The browser is replaced by a client program, which exchanges data with the WebTransactions applications and executes programs in the WebTransactions session. The individual accesses are therefore not linked to a template, but are controlled directly from the client program.

The client program can transfer data to WebTransactions. Depending on the contents, this data is created in the form of new global variables or as attributes of `WT_SYSTEM` or `WT_HOST` in the WebTransactions session. By the same token, the client program can also query data from the addressed WebTransactions session. In addition,embedded or user-defined constructors, methods and functions can be executed in the WebTransactions session. The result of a call such as this is returned to the client program.

The actions, the associated data and the results exchanged between the client program and WebTransactions are coded in an XML-based language. Please refer to the WebTransactions manual “Client APIs for WebTransactions“ for further details on the structure of the messages.

The following diagram shows the basic sequence of the various actions:



3.6 Objects - dynamic data

WebTransactions supports a powerful, yet simple object concept with which the data exchanged between the browser and the host application is modeled at development time, and with which the communication processes are controlled during runtime. WebTransactions structures data using the object hierarchy shown in the diagram below:

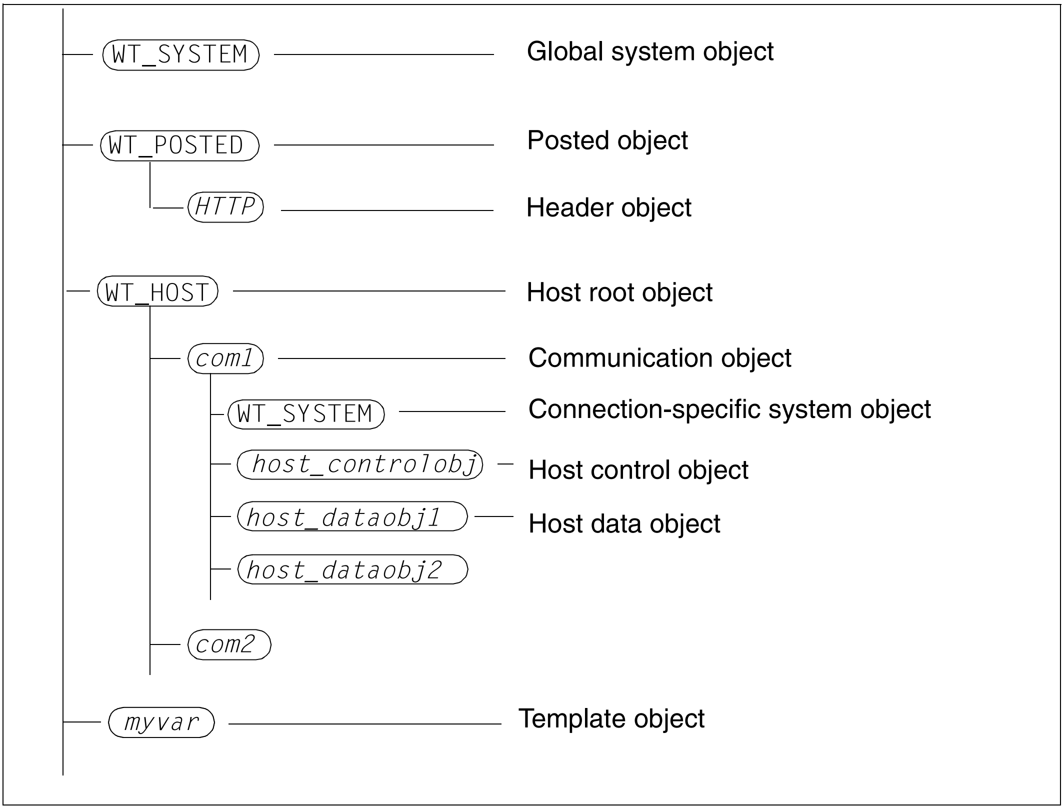


Figure 3: Object hierarchy of WebTransactions

The frames in the graphic show that the structure consists of objects which, in turn, contain other objects. The values given in italics show that the names of objects depend on the specific situation.

During a dialog cycle – see also section “Dialog cycle” on page 62 – the objects defined by WebTransactions are used as follows:

- The global system object (`WT_SYSTEM`) contains attributes that are valid throughout the entire WebTransactions session. WebTransactions uses these attributes, among other things, to manage information about the current session.
- The posted object (`WT_POSTED`) contains the user input.
- The communication objects contain data on the connections that currently exist. Communication objects are used to send data to the host application and receive data from it.
- The host data objects are used for communication between WebTransactions and the host application. They represent the fields of a format of the host application or the data objects (XML structure) of a partner application with which communication takes place over the HTTP adapter. They are created when the `receive` method is called and are transferred to the host application when `send` is called.
- Template objects contain data with a short life which is buffered while a template is being interpreted. They are stored with WTML language resources and functions.

  Additionally, there are objects, which have been defined in module templates (see section “Module templates” on page 52). These objects exist for the entire duration of the WebTransactions session.

The diagram below indicates the individual actions and the utilization of the objects for a synchronized dialog cycle in more detail.

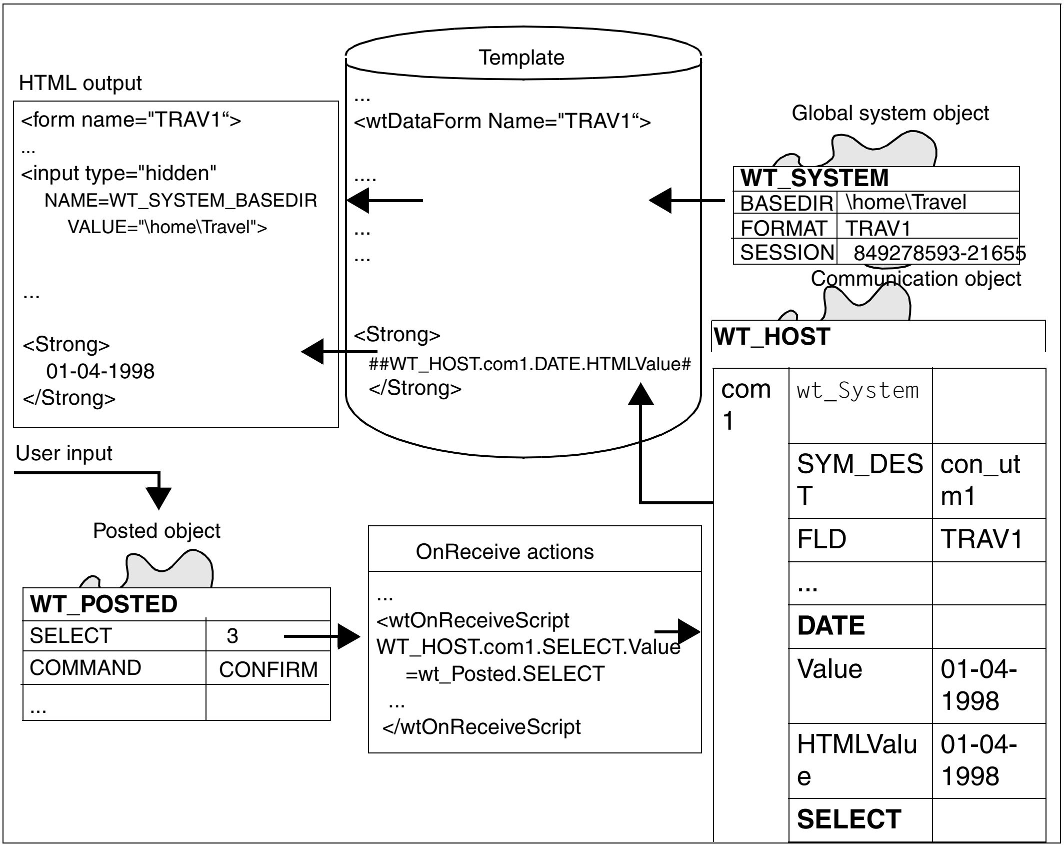


Figure 4: Use of objects in a dialog cycle

The significance of the various objects, their main properties and methods, their interactions and their lifetimes are described in detail in the following sections.

3.6.1 Lifetime of objects

The table below shows the lifetime of the objects predefined by WebTransactions:

Object	**Object lifetime**
Global system object WT_SYSTEM	Duration of the WT session. Attributes created by you continue to exist until the end of the WT session or until they are explicitly deleted.
Posted object WT_POSTED	Duration of the WT session. **synchronized** These attributes contain the data last received from the browser, and continue to exist until the next time data is received from the browser **non-synchronized** These attributes contain the data which was used to call the browser from the non-synchronized template **remote** only available for analysis purposes
Host root object WT_HOST	Duration of the WT session.
Communication object Attribute of WT_HOST	From its creation using the WT_Communication constructor call until the end of the session.
Connection-specific system object Attribute of a communication object	From its creation using the WT_Communication constructor call until its explicit deletion or the end of the session.
Host data object Attribute of a communication object	From its creation using the receive method until the next time this function is called (at which point older host data objects are destroyed) or until close.
Host control object Attribute of a communication object	From its creation using open method until close.
Template objects	**synchronized/non-synchronized** Objects exist from specific creation in the template through to the end of the template processing. **remote** Objects exist from specific creation using methods, constructors or upload accesses through to the end of the communication or until they are deleted explicitly. Objects from module templates The objects exist for the duration of the session or until an explicit deletion occurs.

3.6.2 Object visibility

The visibility of objects and variables varies according to the type of access to WebTransactions. It is dependent on whether the access to the WebTransactions application is synchronized, non-synchronized or carried out via the WT_REMOTE interface. WebTransactions manages the posted and template objects for each type of access separately, in WebLab this is displayed using different object trees:

- The objects created during a synchronized access are displayed in the **synchronous** object tree.
- The objects created during a non-synchronized access are displayed in the **asynchronous** object tree
- The variables created during an access via WT_REMOTE are displayed in the **remote** object tree.

This split management system also affects visibility and, as a result, access to variables:

- A template which executes a synchronized dialog step cannot access the template variables and posted object valid for a non-synchronized step and vice versa.
- Variables with the same name may exist simultaneously in different dialog steps (synchronized, non-synchronized and remote). This variables contain different values since they are managed separately.
- Functions defined in the current synchronized dialog step are not available in a non-synchronized access.

3.6.3 Global system object WT_SYSTEM - session control and long-term data storage

The global system object WT_SYSTEM is created at the beginning of a WebTransactions session and continues to exist for the entire duration of the session. WebTransactions stores information about the current status of the session in this object’s attributes.

In addition, some attributes can be created for the purposes of long-term data storage. These can then be queried in later dialog cycles.

Only one system object exists for each session. Changes in dialog type (synchronized, non-synchronized, remote) are visible even when using the other dialog type.

3.6.3.1 Long-term data storage

Since the system object continues to exist for the entire duration of a session, it is suitable for longer-term data storage. Template objects created using WTScripts exist only for the respective dialog cycle.

For instance, you can buffer a user profile which is entered in an HTML page and evaluated on a later page.

Some system object attributes are created and assigned a value by means of a simple assignment. If an attribute with the selected name already exists, its contents are overwritten.

i To avoid conflicts with predefined attributes, your own attributes should begin with the underscore character (_).

Example

A script creates the attribute `_NEW_ATTRIBUTE` with the value `test`:

```
<wtOnCreateScript>
   WT_SYSTEM._NEW_ATTRIBUTE="test";
</wtOncreateScript>
```

3.6.3.2 Global session control

Information as to which template is to be read next, which language is set for the interface, whether error messages are to be suppressed, etc. is stored in the form of system object attributes. These attributes can be retrieved and in some cases modified, and control the behavior of the WebTransactions kernel.

The following table contains an overview of the attributes and their effect (further attributes are described under the communication interface). Some attributes are set by the kernel at the start (identified as “start”) or can be modified or queried by actions in the templates (identified as “template”) in order to implement specific control functions. Other attributes are sent as hidden fields with each page (identified as “hidden”).

Only those system object attributes that have the same meaning for all WebTransactions protocol variants are described below. Attributes that exist specially for protocol-specific connections or that have a specific meaning for protocol-specific connections are described in the manuals for the supply units in question.

The type of each attribute is shown in the “Use” column.

Attribute	Meaning	Control option	Use
BASEDIR	Base directory for the WebTransactions application	This attribute is set on the start page and cannot be modified thereafter. Value: absolute path name */path/basedir*	hidden start
CGI	CGI environment variables	This attribute is of data type `object`, and contains the specified attributes, which are of data type `string`. It stores the CGI environment values, which are transferred“ from the HTTP server when the CGI program WTPublish.exe is called.	start
AUTH_TYPE		Authentication procedure to be used.	
GATEWAY_INTERFACE		CGI version of the server.	
HTTP_ACCEPT		List of MIME types accepted by the client.	
HTTP_ACCEPT_CHARSET		Character set accepted by the client.	
HTTP_ACCEPT_ENCODING		Browser information indicating the type of documents it can process.	
HTTP_ACCEPT_LANGUAGE		Language accepted by the client.	
HTTP_USER_AGENT		Browser identifier.	
PATH_INFO		Additional path in the requested URL relative to the root directories set in the WWW server.	
PATH_TRANSLATED		Path from PATH_INFO in accordance with the server directory structure to the URL with specifications tagged with?.	
QUERY_STRING		If a CGI script is called when sending an HTML form, this environment variable contains the form data filled out.	
CONTENT_LENGTH		Number of characters transferred when calling the CGI script via the POST method.	
CONTENT_TYPE		MIME type of transferred data when calling via the POST method.	
REFERER_URL		URL from which WebTransactions was started.	
REMOTE_ADDR		IP address of the requesting host.	
REMOTE_HOST		Name of the requesting host.	
REMOTE_IDENT		ID of the user on the requesting host.	
REMOTE_USER		As for `REMOTE_IDENT`. The server determines which of these two variables is set.	

Attribute	Meaning	Control option	Use
REQUEST_METHOD SCRIPT_NAME SERVER_NAME SERVER_PORT SERVER_PROTOCOL SERVER_SOFTWARE		Method of the HTTP request (Post or Get). Virtual path of WTPublish.exe. Name or IP address of the server. No. of the port at which the request arrived. Name of the HTTP server. Protocol version/type.	
CHARSET	Character set specification in the Content-Type field of the HTTP header	If this attribute is set then the content is written as a value of the Content-Type field of the HTTP header in the message generated for the browser. Browsers can set the corresponding character set dynamically. If the attribute is not set then WebTransactions generates the following header: Content-type; text/html; charset=ISO-8859-1 See also attribute HTTP_HEADER CHARSET is overwritten by the attribute HTTP_HEADER	template
COMMUNICATION_ ERROR_FORMAT	Template for error output	If an error occurred in a OnReceiveScript tag with open(), send() or receive() calls, and if this variable is not set to an empty string, the system branches to the specified template. If communication errors occur in a OnCreateScript, this attribute is ignored. These errors can only be detected and handled by querying the ERROR attribute. Values: Empty string: no error output defined. File name of the template *template*[.htm]: error output defined.	template

Attribute	Meaning	Control option	Use
COMMUNICATION_ ERRORS_DISABLED	Switch for deactivating communication error messages	This attribute allows you to control whether or not communication error messages are to be forwarded to the browser/user. In productive mode, you should set this attribute such that the user is not irritated by any error messages that may occur. Errors should be handled in the template, e.g. by querying the *ERROR* attribute or setting the *COMMUNICATION_ERROR_FORMAT* attribute. Values: Empty string: errors are forwarded. Other value: errors are not forwarded	template
DEFAULT_FORMAT	Default for missing template	If the template specified in FORMAT is not found, an attempt is made to read the template specified in DEFAULT_FORMAT.	template
DIALOG_CONTROL_ FORMAT	Control template	This attribute is used in generated templates to suspend the dialog conducted by the host application and return to a “superordinate" template (e.g. menu screen). The template is jumped to using the **Suspend** button.	template
ERROR	Error message	After the execution of the communication functions, WebTransactions stores an error message in this variable if the action failed. If the action was successful, the variable is set to an empty string. The value of the variable can be queried in a template (see also COMMUNICATION_ERROR_FORMAT). A more elegant method of responding to errors during communication is the use of exceptions (see WebTransactions manual “Template Language“)	template
ERROR_LOGFILE	Logfile for errors	If a filename is set for this attribute then WebTransactions does not display any errors. Instead, these are written to the specified file. You must specify the filename without the path. It is created in the base directory.	template

Attribute	**Meaning**	**Control option**	**Use**
`EXIT_SESSION`	End of session	If this attribute is set, WebTransactions ends the session after the next HTML page is sent to the browser. This attribute is defined using the `exitSession()` function. If the `PREVENT_EXIT_SESSION` attribute exists, the WebTransactions session is not terminated even if the *exitSession()* function is executed. See also page 79.	template
`FORMAT`	Name of the next template (synchronized dialog)	In synchronized dialog, this attribute determines the next template to be read. It can be specified in the URL, on the start page, or via WTML tags (assignments) in the template. This attribute is set using the function `setNextPage()` or by means of an explicit assignment. Value: File name of the template *template*`[.htm]` (see "Search strategy" on page 57).	template
`FORMAT_STATE`	State of the current page	This attribute ensures that no page is processed outside the specified sequence. It is automatically set to a value which is unique for each HTML form. This allows the browser history to be used for handling user errors. This attribute cannot be modified.	hidden
`HANDLE`	Name of the current communication object	This attribute is only supported for compatibility reasons with regard to V1.0 and V2.0. As of V3.0, you can name each communication object in order to work with it directly. The value of this attribute sets a communication object to default.	template

Attribute	Meaning	Control option	Use
HREF	Link. Data for creating a link that calls the running WebTransactions session synchronously	This attribute can be used in the template to define a link. It contains the name of the CGI module WTPublish.exe and a value used for page identification (*BASEDIR, SESSION, FORMAT, FORMAT_STATE, LANGUAGE, SIGNATURE, TIMOUT_APPLICATION*). Values: *url_of_webtransactions&name1=value1*&... of the HTML link. `<A HREF=“##WT_SYSTEM.HREF#“> .</A>` calls the current WebTransactions session. See also section “Starting by input of the URL” on page 94.	template

HREF_ASYNC	Header for hyperlink.
	Data for creating a link that calls the running WebTransactions session asynchronously
		This attribute contains the header (script name and name/value pairs of other system variables) for hyperlinks within a session. Unlike HREF, this URL opens WebTransactions for a non-synchronized dialog cycle. The appropriate template must be specified as an additional name/value pair with the name WT_ASYNC_PAGE. An HTLML link in the format
		`<A HREF=`
		`"##WT_SYSTEM.HREF_ASYNC#&WT_ASYNC_`
		`PAGE=myPage">`
		`</A>`
		calls the current WebTransactions session for an asychnronous dialog cycle. If WT_ASYNC_PAGE is not specified, WebTransactions outputs the last page generated synchronously again (this may prove useful, for instance, if frames around dialog pages are to be dynamically loaded or unloaded within a dialog, see example in section “Dialog between WebTransactions and the browser” on page 110).
			template

Attribute	**Meaning**	**Control option**	**Use**
`HTTP_DEFAULT_HEADER`	HTTP header for entire session	If this attribute is set then the content is written to the message generated for the browser as an HTTP header. The attribute must contain all the necessary headers. line breaks (CR LF) and the terminating double line break (CR LF CR LF). This header is written to each generated message as long as `HTTP_DEFAULT_HEADER` is set. WebTransactions no longer generates its own header fields; the HTTP server can, however, create additional HTTP header fields as necessary if these are not contained in `HTTP_DEFAULT_HEADER`.	template
`HTTP_HEADER`	HTTP header	If this attribute is set then the content is written to the message generated for the browser in the form of an HTTP header. The attribute must contain all the necessary headers, line feeds (CR LF) and terminating double line feeds (CR LF CR LF). The attribute `HTTP_HEADER` is deleted after every generated page and must be reset for each page with special headers. If the attribute is not set then WebTransactions generates the following header: `Content-type; text/html; charset=ISO-8859-1` `HTTP_HEADER` overrides the attributes `CHARSET` and `HTTP_DEFAULT_HEADER`.	template
`JAVA_CLASSPATH`	Path of Java classes	The Java class files containing the class methods to be executed in WebTransactions are assumed to be in the subdirectory `java` in the base directory. If the system is to look for them in a different directory, you can set that path with this attribute. Default: *basedir*`/java`	template

Attribute	Meaning	Control option	Use
JAVA_EXCEPTION	Suppress the error messages for unintercepted exceptions	Unintercepted Java exceptions are not passed to the browser/user as error messages. If you define the attribute as a BOOLEAN variable, it is set to false if the method executes successfully. As soon as an exception occurs, this attribute is set to true. Information about the trace file is logged.	template
LANGUAGE	Interface language	This attribute is evaluated as part of the search strategy for the template and the error message file. It is usually set on the start page, but can be modified at any time (e.g. via active user selection). See also section “Subdirectories for style and language variants” on page 56	template
LT_REPLACE_STRING	Replace the HTML tag opening character '<'	This attribute causes the character '<' (less than) to be replaced by the string contained in LT_REPLACE_STRING in all data posted by the browser (attributes of the object WT_POSTED). This prevents HTML tags from taking effect in the contents of data entry fields.	template
MAX_NESTING_LEVEL	Maximum nesting level	With this attribute, you can control the maximum nesting level of templates to be included (<wtInclude>, include(), evaluate()) as well as of function requests. Default value: 99 The forward() function resets the current nesting level to 0.	template
PLATFORM	WebTransactions system environment	Possible values: Windows, UNIX, OSD	start

Attribute	Meaning	Control option	Use
POSTED_UNPARSED	Unprepared data from the browser (query string)	WebTransactions makes the data received from the browser or the query string for a URL that points to WebTransactions available as a WT_POSTED object. If you do not require the posted data in prepared form, e.g. because you want to forward the data to another web server then you can assign this attribute a name which may not appear as a value in the posted data. WebTransactions then creates an object with the corresponding name under the WT_POSTED object. The unprepared data from the browser is then available in this object. Example: WT_SYSTEM.POSTED_UNPARSED="myPost" The myPost object then contains the name/value pair received from the browser.	template
PREVENT_EXIT_SESSION	Keeps the session open despite *exitSession()*	If this global system attribute exists, the WebTransactions session is not terminated even if the *exitSession()* function is executed. This attribute is useful for debugging a session with WebLab. You do not have to disable all the *exitSession()* functions, which may occur at many locations, to prevent termination of the session. This means that you can continue to access the variables and any single-step log of the session that may have been created. When a new session is started, the posted value of WT_SYSTEM_PREVENT_EXIT_SESSION is passed to WT_SYSTEM.PREVENT_EXIT_SESSION. This means that it is not necessary to manipulate the existing templates. The start parameters passed with WebLab are sufficient.	template
PROTOCOL	Name of the host adapter to be used	This attribute is only supported for compatibility reasons regarding Version 1.0. Starting with V2.0, you can attach the protocol to each connection to be opened using the open command. The value of this variable determines which host adapter is used to open a connection if the operand PROTOCOL is not given in the WT_Communication constructor. Values: OSD, MVS, UTMV4	template

Attribute	Meaning	Control option	Use
ROAMING	Re-entry into session	On re-entry into a session, WebTransactions sets the attribute ROAMING to true. This makes it possible, within a template, to determine whether a new login has occurred or whether an existing session is being resumed. In this way, it is possible to re-use the start template for the renewed authentication and, for example, omit a number of initialization operations on resumption by referring to the values in ROAMING. See also section “Roaming Sessions” on page 41.	template
ROAMING_FORMAT	Check authorization to access a roaming session	To prevent unauthorized access to a session, it is necessary to check the authenticity of the user by means of the ROAMING_FORMAT attribute if a session is already running with the specified session ID. To this end, the name of a template must be entered in ROAMING_FORMAT. The authorization to access the session must be checked in this template. See also section “Roaming Sessions” on page 41.	template
SEARCH_HOST_OBJECTS	Searches for variables in host objects	For reasons of compatibility with Version 1.0, WebTransactions also always searches for V2.0 global variables under the default communication object (see HANDLE attribute). This search is not carried out by default as of Version 3.0. If you want to ensure, for compatibility reasons, that this search is carried out, you must set this attribute to "YES". The set mode is valid for an entire dialog step. Any changes made to the attribute only then come into effect for the following dialog steps. Values: "YES", "NO" Default: "NO".	template
SESSION	Session identifier	WebTransactions uses this attribute to find the task (process/thread) assigned to the current session. The attribute is created at the beginning of the session, and is contained in all follow-up pages as a hidden field. The directory below *tmp* which is used to store temporary files assigned to the session has the same name as this attribute. This cannot be modified.	hidden

Attribute	**Meaning**	**Control option**	**Use**
SIGNATURE	Signature	A signature is generated for each HTML page. If a page with an inconsistent signature is received, it is rejected with an error message. The status of the holder task remains unchanged. This attribute cannot be modified.	hidden
STATISTICS	Statistical information about the session	The object STATISTICS contains three numeric attributes. BYTES_SENT contains the number of characters sent by WebTransactions to the browser during the current session, BYTES_RECEIVED contains the number of characters received by the browser. DIALOG_STEPS contains the number of dialog steps. The data is updated automatically at each dialog step.	template
STYLE	Style of the interface	This attribute is evaluated as part of the search strategy for the template. It allows you to select the desired style if several sets of templates are maintained in parallel (e.g. with numerous/few graphics, use of JAVA, etc.) and can be modified at any time. Value: name of the STYLE directory (see section “Search strategy” on page 57).	template
TIMEOUT_APPLICATION	Time in seconds spent waiting for a response from the WebTransactions application	This attribute defines the maximum time spent waiting for a response to a request to WebTransactions before the session is timed out. Value: numerical value Default: 120 seconds.	start template hidden
TIMEOUT_FORMAT	Template executed on session timeout	This attribute specifies the template to be executed on expiry of TIMEOUT_USER before the session is terminated.	start, template
TIMEOUT_USER	Time in seconds spent waiting for a response from the user	This attribute defines the maximum time spent by the browser waiting for a response from the user before the session is timed out. It can be set at the start and modified at any time. Using the setting NOLIMIT the timer for the attribute TIMEOUT_USER is not activated. Value: numeric value or NOLIMIT. Default: 600 seconds.	start template

Attribute	Meaning	Control option	Use
`WTML_VERSION`	Versions specification for WTScript	The introduction of version 1.2 of the JavaScript language has changed the behavior of some methods in the installed classes. This improved behavior should also be available in WebTransactions. If you want to set the old behavior for these methods, you must set the value to 2.0. Values: 7.5, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0 Default value: 7.5	template
`WWWDOCS_VIRTUAL`	Virtual path for the wwwdocs directory and base directory	This attribute is required in order to address resources in the template which are located in the `wwwdocs` directory. It is read from the administration specifications at runtime and cannot subsequently be changed during the session.	hidden, start

3.6.4 Posted object WT_POSTED - data from the browser

The posted object stores data sent by the browser. It is updated in the dialog cycle each time browser data is received. Both synchronized and non-synchronized dialogs (with the browser) begin by updating the posted object. The data of a posted object is always visible during the current dialog step and is no longer there for subsequent steps.

The attributes of the posted object and their values correspond to the name/value pairs of the HTML elements that are returned by the browser, where *name* is the HTML page object/field manipulated by the user and *value* is the resulting value to be processed. The attributes can be addressed using the name under which the object/field was defined on the HTML page.

The posted object is a global variable of data type `object`, and its attributes are of type `string`. If the message contains several name/value pairs with the same name, an array is created with this name. The entries are of type `string`.
The toString method (`WT_POSTED.`*arrayname*`.toString()`) returns the first attribute.

Example

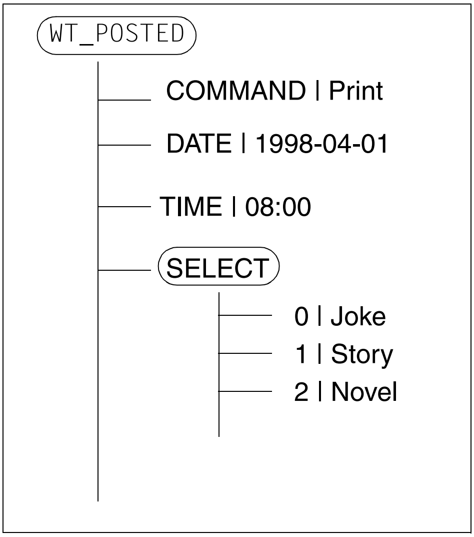
An HTML page contains the following tags:

```
<input type="text" name="DATE" value="1998-04-01" >
<input type="text" name="TIME" value="08:00" >
<input type="submit" name="COMMAND" value="Print">
<select name="SELECT" multiple>
<option value="Drama">Drama
<option value="Story" selected>Story
<option value="Joke" selected>Joke
<option value="Novel" selected>Novel
<option value="Poem">Poem
</select>
```

The following name/ value pairs are sent by the browser:

```
COMMAND=Print
TIME=08:00
DATE=1998-04-01
SELECT=Joke
SELECT=Story
SELECT=Novel
```

This results in the following object hierarchy:



In the diagram above, objects that themselves contain objects are surrounded by a frame.

The posted object can only be queried when executing onCreate scripts (non-synchronized) or onReceive scripts and when executing the onCreate script (synchronized) of the next template. Value assignments are ignored.

The posted object contains the object `WT_POSTED.HTTP`, under which all HTTP headers of the respective request are created as attributes. These attributes are also read-only. To edit the fields in the HTTP header of the response, you must use the system object attributes `WT_SYSTEM.HTTP_HEADER` (see page 77) and `WT_SYSTEM.HTTP_DEFAULT_HEADER` (see page 77).

By interpreting this browser data as an object, it is possible to preserve a uniform syntax in the template for access to dynamic data.

Example

If a page contains the following three buttons:

```
<Input Type="SUBMIT" Name="CHOICE" Value="Arts">
<Input Type="SUBMIT" Name="CHOICE" Value="Recreation">
<Input Type="SUBMIT" Name="CHOICE" Value="Sport">
```

you can evaluate the values returned by the browser in the next template as follows:

```
You have opted for ##WT_POSTED.CHOICE#.
```

When WebTransactions is accessed via the `WT_REMOTE` interface, the posted object is supported for analysis purposes. In WebLab, you can see the control and data part of the posted object for analysis purposes. However, it is not recommended that you access this data using methods or constructors.

Special attributes of `WT_POSTED`

File upload

If a POST request arrives at WebTransactions whose content type starts with `text/` , the content of the body is stored in the `BODY` attribute of the `WT_POSTED` object, where it is available for further processing. The body may only contain printable characters, otherwise the string is regarded as terminated at the first binary zero.

Other special attributes

If you specify `/startup` when starting a session, the posted values of some system object attributes `WT_SYSTEM_`*`XXX`* are stored under `WT_SYSTEM.`*`XXX`*. See also the sections “Start options” on page 93 and “Subdirectories for style and language variants” on page 56. This concerns the following attributes:

– `BASEDIR`
– `FORMAT`

- LANGUAGE
- PREVENT_EXIT_SESSION
- STYLE

3.6.5 Host root object WT_HOST - managing connections to host applications

All objects that contain information on the connections to host applications opened during a WebTransactions session are combined under WT_HOST. A separate communication object is created for each connection, and contains the data (host data objects, host control objects, connection-specific system object) relating to this connection.

During a WebTransactions session, it is possible to work simultaneously with several open connections to different host applications and host application types.

There is only one host root object and one communication object per session. Changes in dialog type (synchronized, non-synchronized, remote) are also visible during the other dialog types.

3.6.5.1 Host communication object WT_HOST.*Comobj* - managing a host connection

Because, in each session, multiple connections to different host applications an be opened either sequentially or in parallel, WebTransactions (and the template programmer) must be able to distinguish clearly between the different connections. Therefore, a communication object is generated (explicitly or implicitly for each connection). The names of these communication objects are freely definable.

For the sake of clarity, WebTransactions generates all communication objects under a common root, the host root object WT_HOST.

A host communication object contains the following:

- internal management information, including information on the host adapter used by the connection. This information cannot be accessed.
- the current host data objects and host control objects
- the connection-specific system object, if any. The attributes of this object are dependent on the host adapter and are described in the corresponding manuals as attributes of the system object.

3.6.5.2 Host data objects - host application data

Host data objects are provided for the transfer of data between WebTransactions and the host application. These correspond to the host application data and store the individual fields of a screen format. They are created after a host message is received, and are stored as attributes of the respective communication object. They are then available in the form of a screen image. They and can be read, overwritten, or transferred, and are destroyed each time a new host message is received.

For display in the browser, the host data objects are incorporated in the HTML page. All generated templates are constructed in such a way that the input fields are mapped to the corresponding HTML interface elements (for example, `<Input Type="text" ...>`), while output fields are mapped to HTML text.

Data is sent to and received from the host application by means of the `send` and `receive` methods of the class `WTCommunication` which are present in the WTScript language scope. This can occur when the HTML page is created (on Create) or after data is received from the browser (on Receive). The lifetime of a host object is therefore not linked to the dialog cycle. Several generations of host objects may appear and disappear again during a dialog cycle, or even during a particular phase of the dialog cycle (HTML generation or execution of a receive method), or one particular generation may extend over several dialog cycles.

Host objects exist only if a connection to the host application is open (see WebTransactions manual “Template Language“, open method).

Since the host adapters create host objects under the corresponding communication object, several objects with the same name can exist in parallel. However, their names are unique by virtue of the fact that you address host objects by their fully qualified name, i.e. using the path to the communication object (`WT_HOST.`*Comobj*`.Host_object.attribute`).

The precise properties and names of host objects depend on the host adapter used, and are described in the manual for the relevant host connection.

3.6.5.3 Host control objects - management data for a format

Host control objects are provided for controlling the host connection, and continue to exist for the entire lifetime of the connection. They supply information not only on an individual field, but on the entire screen format. For instance, this may indicate the field in which the cursor is positioned and the layout of fields on your screen.

The precise properties and names of host control objects depend on the host adapter used, and are described in the manual for the relevant host connection.

3.6.5.4 Connection-specific system object WT_Host.*Comobj*.WT_SYSTEM - connection-specific control functions

The attributes of the connection-specific system object control the connection to a host application. They are specific to the host adapter used, and are described in the corresponding manuals.

The connection-specific system object is created when the communication object is created using the constructor of class `WT_Communication`. As it is a perfectly normal object, it can be deleted within a WTScript using the `delete` operator.

If the connection-specific system object exists, WebTransactions only accesses the attributes of this system object to control the connection. Attributes with the same name belonging to the global system object are ignored.

3.6.5.5 WTScript and communication objects

The diagram below presents an example of how to use the WTScript statements to establish a connection to a host application and describes the objects created by WebTransactions in the process:

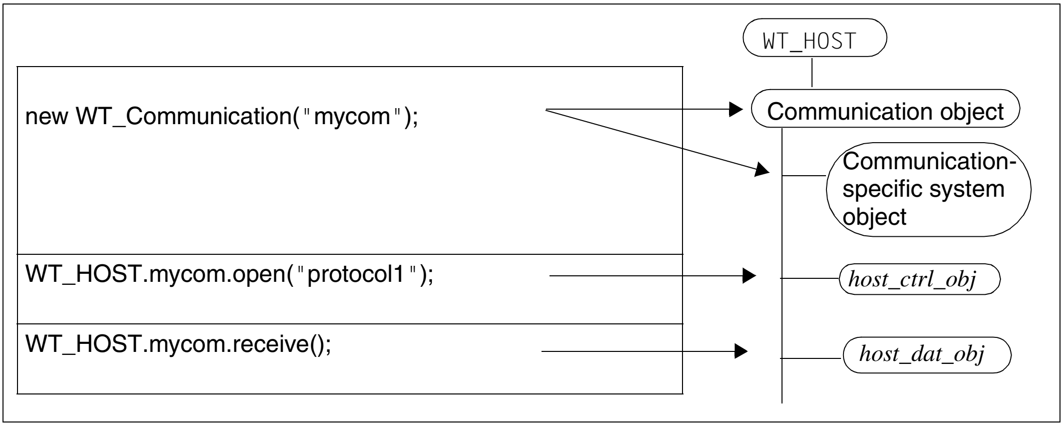

WT_HOST
new WT_Communication("mycom");
Communication object
Communication-specific system object
WT_HOST.mycom.open("protocol1");
host_ctrl_obj
WT_HOST.mycom.receive();
host_dat_obj

Figure 5: Interaction of WTML and communication objects

The `WT_Communication` constructor call creates a communication object under `WT_HOST` and, under this, a connection-specific system object. You also specify the name of the communication object in this call.

You call all further actions involving this connection as methods at this communication object.

A connection is opened using the `open` method call. In these calls, you define the host adapter with which the connection was established. The host control objects are also created with this call.

Data is received from the host application using the `receive` call. The host data objects are also created with these calls.

3.6.6 Template objects – Short-term intermediate data storage

Template objects (also known as template variables) offer the template programmer a short-term storage area for data to be buffered in WTScript areas, WTML tags, or evaluation operators of a template.

They can be defined anywhere in these areas, and continue to exist from the time they are defined until the end of the dialog step (synchronized or non-synchronized), until the end of the session (remote) they are explicitly deleted. The template objects of the various dialog types (synchronized, non-synchronized and remote) are managed separately and have no effect on each other.

Template objects are created simply by using a name (identifier that does not correspond to any predefined WTML object) in a statement or assignment, and can include all variable types available in WebTransactions (see WebTransactions manual “Template Language“).

Example

```
<wtonCreateScript>                         (1)
a = 5;
b = a + 1;                                 (2)
</wtonCreateScript>
##a# is one less then ##b#                 (3)
```

(1) In the script area, a template object `a` is created and assigned the value 5.

(2) The value 1 is added to the value of `a` in the evaluation operator and the result is copied to the template variable `b` thereby created.

(3) In the HTML area, the evaluation operators are used to replace the template objects `a` and `b` with current values.

When you execute these statements with WebTransactions, the HTML page contains the text:

```
5 is one less than 6
```

Template objects can also be used to contain further objects/attributes.

For instance, the WTScript given below creates the template object `x` and assigns it a new attribute `y` with the value 1:

```
  <wtonCreateScript>
  x = new Object();
  x.y = 1;
  </wtonCreateScript>
```

4 Execution of a WebTransactions application

A WebTransactions application is an intermediary between dialog applications on a host and the graphical interface in the browser. It is, however, also possible to access a WebTransactions application via a client program and the `WT_REMOTE` interface. This chapter describes the two interfaces used to access WebTransactions.

This chapter focuses on work with the dialog interface and the control options for the template programmer. The description of the dialog interface includes

- the start options for a WebTransactions application: direct input of the URL, link or form. The inputs that can be processed on the start page are also described.
- the exchange of data during the session: the HTML form and HTML link
- the two control options of a WebTransactions application: passive via the host application or active via the template
- the synchronized and non-synchronized dialog
- terminating a session
- diagnostic capabilities
- the transfer of a WebTransactions application

This is followed by a brief presentation of the `WT_REMOTE` client interface.

Finally, you will find a description of how you can administer a WebTransactions application from the browser.

4.1 Creating a WebTransactions application

You now use the following procedure to link your host application to the Web:

1. Install the appropriate WebTransactions supply unit on your integration server. The installation procedure for each supply unit is described in the relevant manual.
2. Each WebTransactions supply unit is supplied with an administration program and the WebLab development environment with which you can carry out all subsequent stages of the integration. The administration program is described in chapter “WebTransactions server” on page 131.
3. Start WebLab using the command **Start/Programs/WebTransactions 7.5/WebLab**. The main window of WebLab is displayed on the screen.
4. When you begin working with WebLab for the first time you must specify the browser you wish to use, to do this use the command **Options/Preferences/Programs**.
5. Select the command **Administration/Server**, to start the administration program.
6. Log in to the administration program as the user `admin`. When you call the administration program for the first time, assign a password to the user `admin`.
7. Request the required number of licenses and enter these.
8. Set up a WebTransactions user for each developer who is supposed to edit WebTransactions applications.
9. Create a pool for your base directories and assign access authorization for this pool to the user `admin` and possibly to other users.
10. Save the new configuration.
11. Exit the administration program and close the browser.
12. In WebLab, create a new project and a new base directory for your host application using the command **Project/New**. In the **Create Base Directory** dialog box, select one or more of the host adapters you are offered.

13. Create an individual start template. Enter the data for your host application in the **connection parameter** tab.

    If you want to connect a host application to WebTransactions for openUTM, you must also perform the following steps:

    – Log onto the host computer under the ID under which the host application you wish to integrate is located.

    – Use the `IFG2FLD` tool from the IFG/FHS library to create the format description sources from which WebLab is to create templates. `IFG2FLD` is described in the WebTransactions manual “Connection to openUTM Applications via UPIC”.

    – Transfer the format description sources in text mode to the computer on which WebLab is running.

    – In WebLab select the command **Generate/Templates/from IFG library** to generate templates for conversion from the format description sources.

14. Use the command **File/Start Session** with the parameters in the dialog field **Start Session** to create a link via WebTransactions to the host application. Enter the individually created start template as the start template here. The parameters of the dialog box **Start Session** are described in section “Starting a session” on page 161 and in the online help system.

    When you confirm this dialog box, the specified browser is started at the development host. The individual start template establishes the connection to the host application and displays the first format in the browser. Depending on the configuration of the host application, this may immediately be the host application’s start format (e.g. openUTM) or an emulation display (e.g. in the case of `$Dialog`) in which you then start the host application.

By completing these steps, you have not only linked your host application to the Web, you have also created and started a WebTransactions application.

For a detailed discussion of this topic, see chapter “The WebLab development environment” on page 155.

4.2 Starting a WebTransactions dialog application

You can start a WebTransactions dialog application by means of direct input of a URL or via a start page in the browser. On this start page, a link or a form points to the first template of the WebTransactions application, the so-called start template. It is the start template that actually starts the WebTransactions application. The diagram below shows how a WebTransactions application is started from a start page.

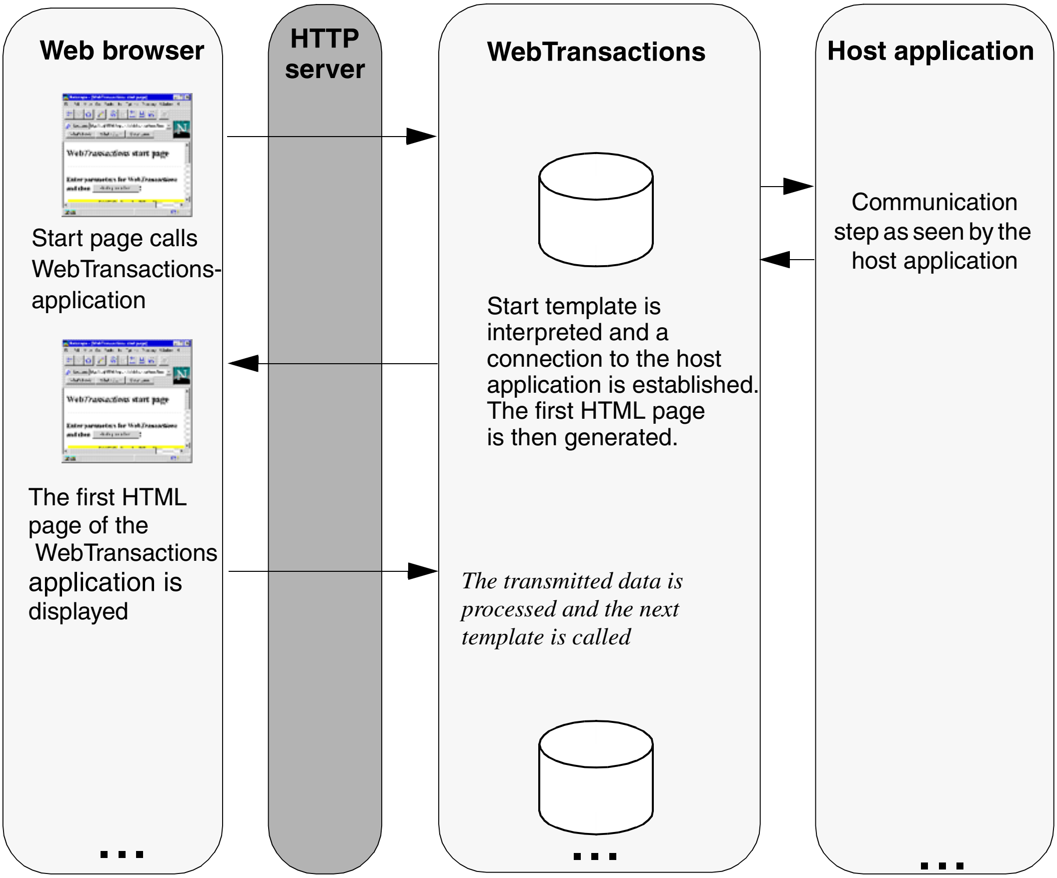

Web browser
HTTP server
WebTransactions
Host application
Start page calls WebTransactions-application
Start template is interpreted and a connection to the host application is established. The first HTML page is then generated.
Communication step as seen by the host application
The first HTML page of the WebTransactions application is displayed
The transmitted data is processed and the next template is called
...
...
...

Figure 6: Starting a WebTransactions application from a start page

WebLab provides a WTBean to support you in creating a start template. You must create the start page the user sees in the browser yourself. In addition to defining the base directory and start template for the WebTransactions session, you can also enter additional data at the browser before the session begins, so that WebTransactions has this information to hand from the outset

i Please note that the start page for a WebTransactions application must be located in the web server's document directory so that it can be found by the web server. You are recommended to save the start page in the `wwwdocs/html` directory. This has the advantage that the file is taken into consideration when the WebTransactions application is transferred and that the paths for the associated program calls (wtPublish, wtCluster) are adapted as required.

4.2.1 Start options

Each session begins with user input at the browser. There are various options:

- Specifying the URL address of a WebTransactions application in the browser or activating a link
- Completing and submitting a form
- Starting via the interface `WT_REMOTE`

The first two start methods can be simulated with WebLab: in this case, the start specifications are not made in the web browser but in WebLab.

On all platforms supported by WebTransactions, the CGI interface to the HTTP server is also supported. The CGI program that you specify in the URL is named `WTPublish.exe`. For Windows platforms, a link module for the ISAPI interface (`WTPublishISAPI.dll`) is also supplied. You can use this on HTTP servers that support the ISAPI interface (e.g. MS Internet Information Server). Since ISAPI brings considerably improved performance compared to CGI, it is advisable to use ISAPI.

If you use the ISAPI interface to WebTransactions, you must replace `WTPublish.exe` by `WTPublishISAPI.dll` in the corresponding URL.

4.2.1.1 Starting by input of the URL

The URL used to call WebTransactions has the following format:

`http[s]://`*machine*`/`*cgiPath*`/WTPublish.exe/`*basedir*`?`*startTemplate*`[.htm]`

It starts WebTransactions on the specified machine in the corresponding base directory.

machine
: is the Internet address or the symbolic name of the host on which WebTransactions is installed (with the port number for the HTTP server if applicable).

cgiPath
: is the path (prefix) for CGI programs defined for the HTTP server there.

basedir
: is the base directory under which the WebTransactions application is installed. It is an absolute path name (and includes the drive name under Windows).

startTemplate
: specifies the first template to be executed. The suffix `htm` can be omitted. To generate the first page, the template *startTemplate* in the style `forms` is used, i.e. the file:

  basedir`/config/forms/`*startTemplate*`[.htm]`

With this method, no further user data can be sent to WebTransactions in the first step.

This URL can be located directly in the browser or in a link, for example to a called page:

`<a href="http[s]://`*machine/cgiPath/*`WTPublish.exe`*/basedir*`?`*startTemplate*`[.htm]">`
Text or image for starting WebTransactions`</a>`

Starting with additional values

If you want to specify additional values when starting, you must construct the URL as follows:

`http[s]://`*machine*`]/`*cgiPath*`/WTPublish.exe/startup?`
`WT_SYSTEM_BASEDIR=`*basedir*`&WT_SYSTEM_FORMAT=`*startTemplate*`[.htm]`
`&myData=data...`

Unlike the previous methods, the base directory and start template are specified as name/value pairs. You must use `WT_SYSTEM_BASEDIR` and `WT_SYSTEM_FORMAT` as names. With this method, further name/value pairs can be entered in `QUERY_STRING`. These are then made available by WebTransactions in the first dialog cycle as attributes of the posted object (`WT_POSTED`).

The names of the attributes correspond to the names of the name/value pairs. For example, if your URL contains a pair `&User=Person`, then you can access the value of this pair in the start template via `WT_POSTED.User`.

Starting in another style or language

Starting a session in another language or another style represent special cases of starting with additional values. Both cases can be defined via specific name/value pairs.

```
http[s]://machine/cgiPath/WTPublish.exe/startup/WT_SYSTEM_BASEDIR=
/basedir?WT_SYSTEM_FORMAT=startTemplate&WT_SYSTEM_LANGUAGE=language&
WT_SYSTEM_STYLE=style
```

Here, the name of the preferred style directory is `WT_SYSTEM_STYLE`, and that of the preferred language directory `WT_SYSTEM_LANGUAGE`.

For a detailed description of how WebTransactions searches for the employed templates, see section “Subdirectories for style and language variants” on page 56.

When you start a session in this way, the start template is searched for in the specified style and/or the specified language.

Starting at a cluster member

A session at a WebTransactions cluster is started using the following link. See also section “Cluster concept” on page 142:

```
http[s]://machine/cgiPath/WTCluster.exe/cluster-id[?WT_SYSTEM_FORMAT
=startTemplate[&param2....]]
```

4.2.1.2 Starting using an HTML form

An HTML form used to call WebTransactions has the following format:

```
<FORM METHOD="POST"
ACTION="[http[s]://machine]/cgiPath/WTPublish.exe/basedir?startTemplate[.htm]">
   additional text and data
</FORM>
```

machine

is the Internet address or the symbolic name of the host on which WebTransactions is installed (with the port number for the HTTP server if applicable). This part is added by the browser and can therefore be omitted if the start page containing this link was loaded from the same machine.

cgiPath

is the path (prefix) for CGI programs defined for the HTTP server there.

basedir

is the base directory under which the WebTransactions application is installed. It is an absolute path name (and includes the drive name under Windows).

startTemplate

specifies the first template to be executed. The suffix `htm` can be omitted. To generate the first page, the template *startTemplate* in the style `forms` is used, i.e. the file:

basedir`/config/forms/`*startTemplate*`[.htm]`

Within the form, it is possible to specify further HTML input elements (e.g. input fields, selection lists), which are then sent to WebTransactions in the first dialog cycle when this method is used. This data is made available as attributes of the `WT_POSTED` object when processing the start template.

The names of the attributes correspond to the names of the input elements. If, for example, your form contains an input field `<input type=text name=User>` you can access the value of this field in the start template via `WT_POSTED.User`.

Starting in another style or language

If you want to start your WebTransactions application with a start template belonging to another style or another language then you should use the following form:

```
<FORM METHOD="POST"
ACTION="[http[s]://machine]/cgiPath/WTPublish.exe/startup]">
<input type="hidden" name="WT_SYSTEM_FORMAT" value="startTemplate">
<input type="hidden" name="WT_SYSTEM_BASEDIR" value="basedir">
<input type="hidden" name="WT_SYSTEM_STYLE" value="mystyle">
<input type="hidden" name="WT_SYSTEM_LANGUAGE" value="mylanguage">
additional text and data
</FORM>
```

Unlike in the preceding method, the base directory and start template are specified in hidden input fields. You should use `WT_SYSTEM_BASEDIR` and `WT_SYSTEM_FORMAT` as the names of the input fields.

Starting at a cluster member

To start a cluster session from a form you must use the `GET` method. See also section “Cluster concept” on page 142:

```
<form method="get"
      action="http[s]://machine/cgiPath/WTCluster.exe/cluster-id">
      [<input type=“hidden“ name=“WT_SYSTEM_FORMAT“ VALUE="startTemplate">]
additional text and data
</form>
```

4.2.1.3 Starting with WT_REMOTE

A client program starts a session for exclusive use via the interface `WT_REMOTE` using the method `START_SESSION`.

Client programming is described in detail in the WebTransactions manual “Client APIs for WebTransactions” .

4.2.2 Templates at start time

At start time, you can either use the supplied start templates or, alternatively, you may generate application-specific start templates. This section describes the supplied start templates and indicates the ways in which you can branch from the individual start templates.

4.2.2.1 General start template wtstart.htm

Special start templates are supplied to help you test your WebTransactions application. You can use these start templates to set parameters for the host applications and start any number of host applications in parallel.

The start template `wtstart.htm` supplied by WebTransactions allows you to

– establish a connection to any host application

– establish several simultaneous connections to several host applications

On this HTML page, you define the global parameters for your WebTransactions session. When opening a connection, the system branches to the connection-specific start page in which you enter the parameters specific to this connection.

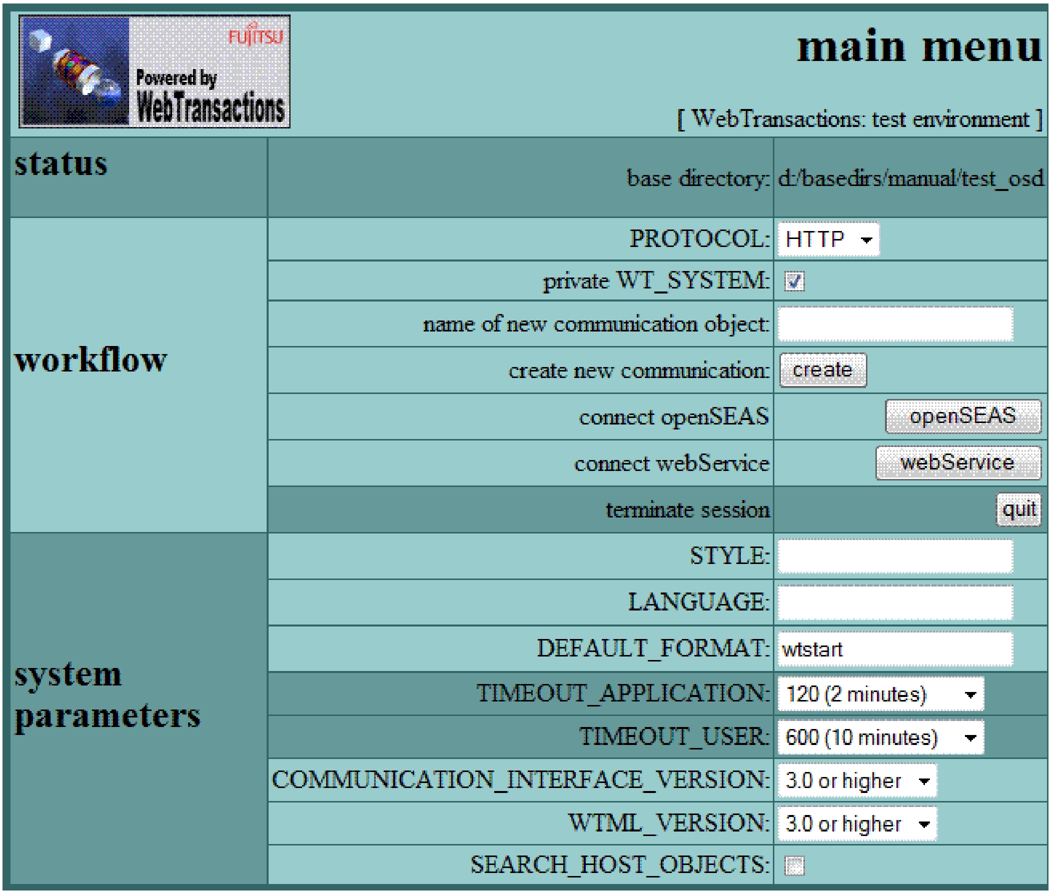
i In a productive application, you will generally create your own start template which automatically determines the most important information for your WebTransactions application and requires minimum user input. WebLab provides a WTBean to help you during the creation of a start template.

If you are working with several simultaneous connections, you should refer to the examples on integrating the application (see also section “Interaction between start templates when integrating the application” on page 103) for information on how to proceed.

i The UTMV4 host adapter does not support the simultaneous operation of two connections. However, you can establish parallel connections to openUTM applications via the OSD host adapter.

When creating a base directory, `wtstart.htm` is stored in the subdirectory `config\forms`. This start template can be specified at the beginning of a WebTransactions session.

The diagram below shows the HTML interface of `wtstart.htm`:



main menu

[WebTransactions: test environment]

status	base directory:	d:/basedirs/manual/test_osd
workflow	PROTOCOL:	HTTP ▾
	private WT_SYSTEM:	☑
	name of new communication object:	
	create new communication:	create
	connect openSEAS	openSEAS
	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:	□

All the system parameters are attributes of the system object and are described in section "Global system object WT_SYSTEM - session control and long-term data storage" on page 70.

PROTOCOL
Specify the type of host application or communication module.

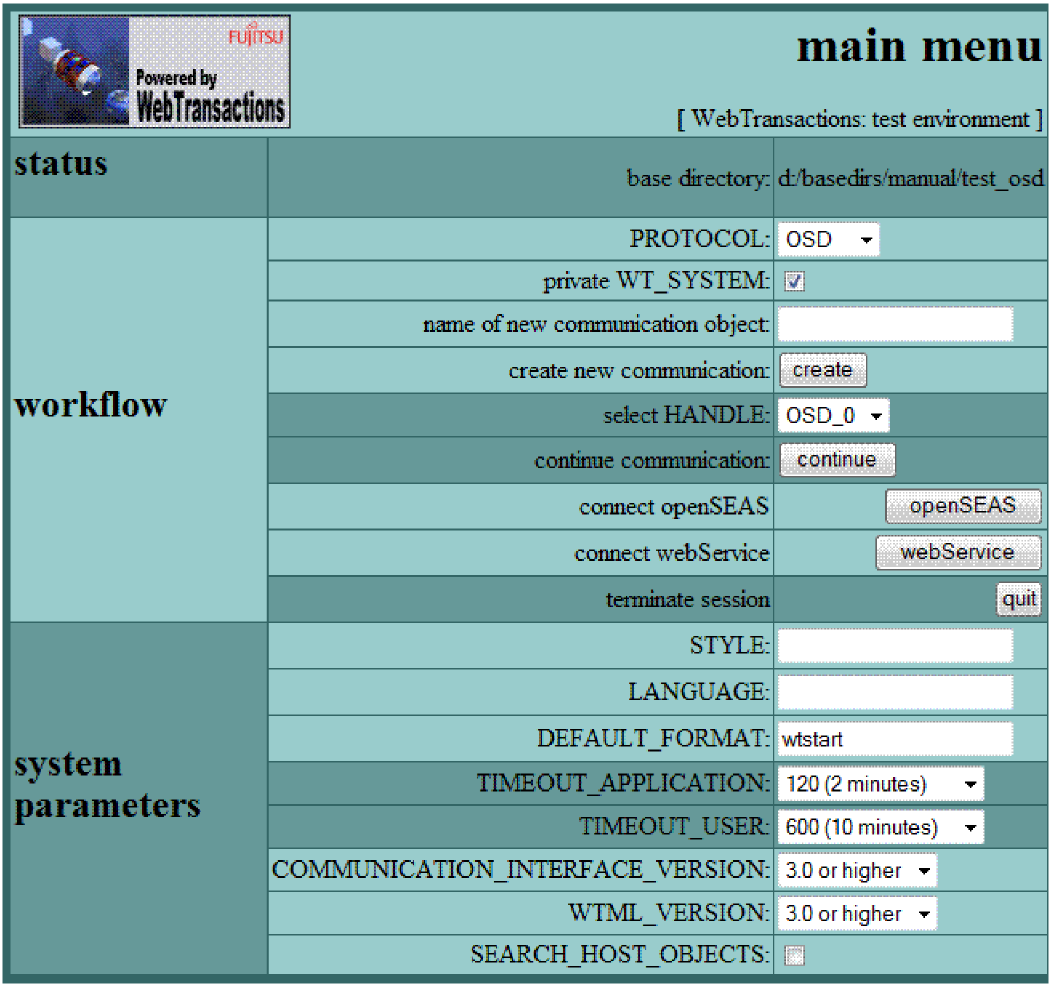
private WT_SYSTEM
The option of a connection-specific system object under the communication object is activated by default.

name of new communication object
You can assign a name for each communication object and access the communication object in the template under this name. The name you specify here must match the name of the communication object in the templates used.

connect webService
Use this button if you want to start a Web frontend for a Web service. The template `wtconnectWebService.htm` is displayed in the browser. Here you can select the templates generated for Web services individually.

create new communication
Use this button to create a new communication object. The protocol-specific start template (`wtstart*.htm`) is displayed in the browser to allow you to enter the parameters for the connection to the host. The `*` character in `wtstart*.htm` stands for the corresponding host adapter. If you then click on the **goto main menu** button in the `wtstart*.htm` template, `wtstart.htm` is displayed again, thus allowing you to create a further communication object and, consequently, a further parallel connection.

If you have already started a connection and interrupt this connection with the **Suspend** button, for instance, a slightly modified `wtstart.htm` is output. In this case also, you can create additional communication objects.



main menu

[WebTransactions: test environment]

status	base directory:	d:/basedirs/manual/test_osd
workflow	PROTOCOL:	OSD
	private WT_SYSTEM:	☑
	name of new communication object:	
	create new communication:	create
	select HANDLE:	OSD_0
	continue communication:	continue
	connect openSEAS	openSEAS
	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:	☐

select HANDLE

From the list displayed, select the name of a communication object (HANDLE) for an existing open connection to a host application.

continue communication

Continue the connection selected under `select HANDLE`.

`wtstart.htm` branches to a connection-specific start template if a connection is established or the dialog with a connection is continued.

4.2.2.2 Connection-specific start templates

The following connection-specific start templates exist:

- `wtstartHTTP.htm`
- `wtstartUTMV4.htm`
- `wtstartOSD.htm`
- `wtstartMVS.htm`
- `wtconnectOpenSEAS.htm`
- `wtconnectWebServices.htm`

When creating a base directory, these start templates are stored in the subdirectory `config\forms`. These specific start templates instigate dialogs with the host application. They are described in the manuals for the relevant communication modules. `wtstart.htm` branches to one of these start templates when a connection is established or dialog with a connection is resumed.

4.2.2.3 Interaction between start templates when integrating the application

When you call a WebTransactions application on the start page, a WebTransactions session is started (see section “Starting a WebTransactions dialog application” on page 92). The following diagram illustrates the various branch options available in the individual start templates:

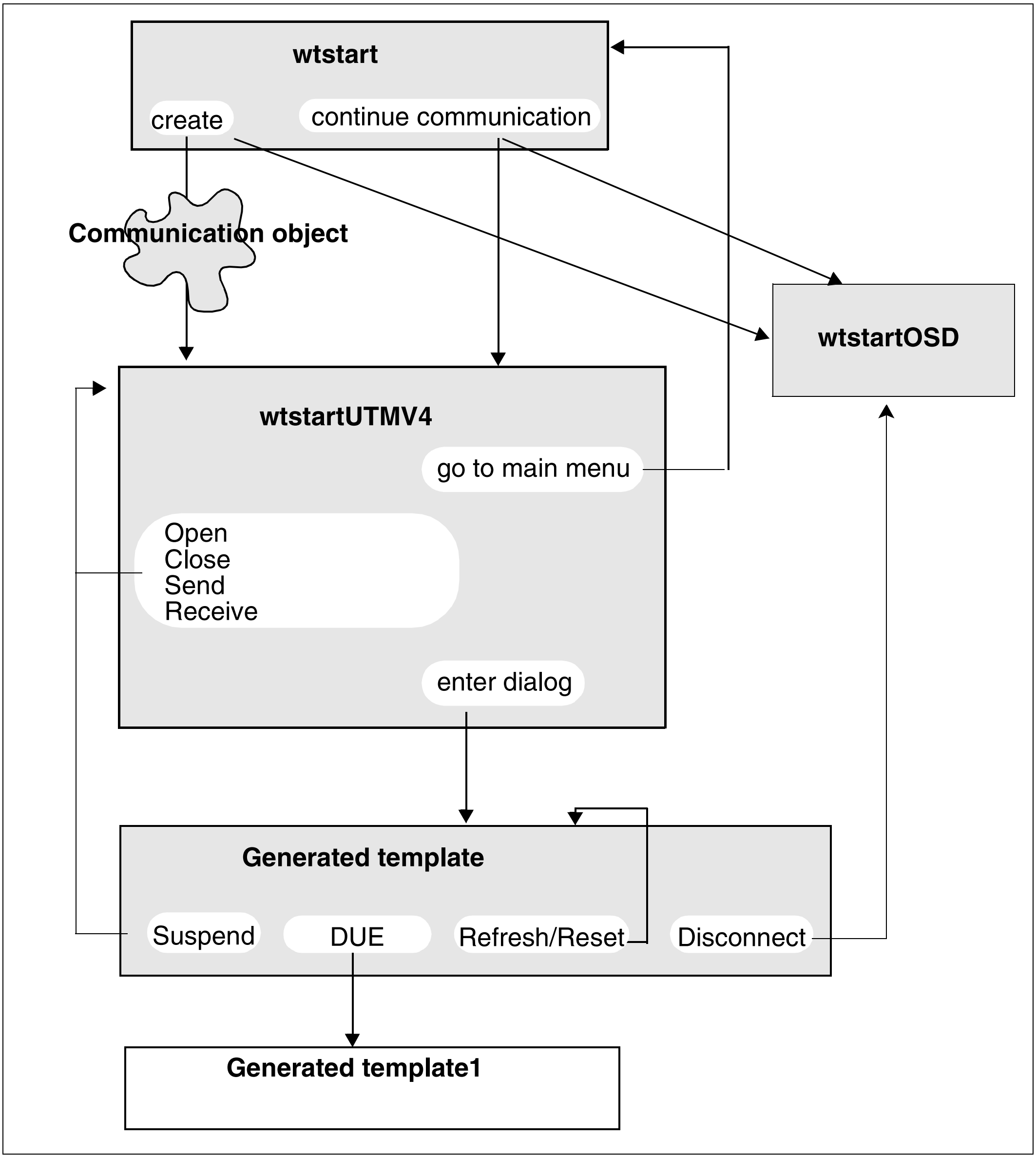


Figure 7: Branch options in the start templates

`wtstart.htm` creates a communication object `WT_HOST.`*Comobj* for various host applications. This is achieved by selecting the **create** button, which initiates the constructor call of the communication object `new wt_Communication` (see WebTransactions manual “Template Language”).

If you return to the general start template, you can use the **continue communication** button to continue communication on other existing connections. `wt_System.FORMAT` is then set to `wtstart*.htm`, and the connection-specific start template is output.

In the connection-specific start templates (see manuals for the individual host adapters), you can return to the general start template using **go to main menu**. You must select this option if you wish to establish a connection to a further host application. You can switch to existing connections directly by means of **go to** ***<commobj>***.

Furthermore, it is possible to initiate actions with the connection (open the connection, send data, receive data...). In particular, you can begin a dialog with the connection. With this option, you use a template of the corresponding host application. If you are working with WebTransactions generated templates, these pages have a button bar containing the key assignments.

In addition to the normal key assignments, the following buttons are available in the button bar:

- **Reset** for resetting the input fields. In this case, no contact is made with the host application.
- **Refresh** to refresh the screen display.
- **Disconnect** to terminate the connection to the host application.
- **Suspend** to interrupt the dialog with a host application (only if the attribute `WT_SYSTEM.DIALOG_CONTROL_FORMAT` is present).

4.3 Data exchange during the session

After a WebTransactions application is started, you make various elements for input, selection options, etc. available to the user in the browser. The values of your responses must be returned by the browser to the WebTransactions session. HTML offers two options for this purpose: the FORM tag and the HTML link.

4.3.1 FORM tag

The HTML tags `<FORM>` and `</FORM>` enclose an area which is used for the input data. Within this area, the values of dialog elements (input fields, pick lists, etc.) are combined and sent from the browser to a program. In WebTransactions, this option is used by the start page, for example. In the subsequent dialog cycles, the WTML tags `<wtDataform>` and `</wtDataform>` are used for this area in the template (see the WebTransactions manual “Template Language”). You no longer need to specify the URL of the processing CGI program. WebTransactions generates a FORM tag which addresses the current WebTransactions session from a `wtDataform` tag on the generated page.

Example

```
<wtDataform>
     Enter your address:
     Street: <input type=“text“ name=“STREET“> <br>
     City:   <input type=“text“ name=“CITY“> <br>

</wtDataform>

<wtOnReceiveScript>
     hostapp.STREET.VALUE=WT_POSTED.STREET;
     hostapp.CITY.VALUE=WT_POSTED.CITY;
</wtOnReceiveScript>
```

4.3.2 HTML link

If you wish to access WebTransactions via a link, you must specify the URL of the program to be called yourself, and attach the user input to the URL in the form of name/value pairs.

The HREF attribute of the global system object contains the URL of the WebTransactions CGI program `WTPublish.exe`. All name/value pairs required by WebTransactions to identify the current session are already attached to this URL.

All that remains is to add the name/value pairs corresponding to the user input to the URL. In other words, a link is defined as follows:

```
<A HREF="##WT_SYSTEM.HREF#&name1=value1&name2=value2&...">Link</A>
```

`WT_SYSTEM.HREF`
This system object attribute contains the URL of the CGI program `WTPublish.exe` and, in encrypted form, the hidden fields on the current page (BASEDIR, FORMAT, FORMAT_STATE, LANGUAGE, SESSION, SIGNATURE, TIMEOUT_APPLICATION).

`name/value`
These name/value pairs are used to define the additional values sent when the link is activated. If the link is activated by the user, these pairs are available in the posted object. They must not contain any blanks or the characters & and =. The following characters should be used instead:

Characters not permitted	**Replace with**
%	%25
&	%26
=	%3D
+	%2B
Blanks	%20
/	%2F
:	%3A

Links enable you to jump to other processing steps in a WebTransactions application (e.g. return to main menu) and are often used to improve appearance.

It is also possible to offer the user a selection of links which can be activated, e.g. instead of a drop-down list.

Example

A selection of links; the posted data is evaluated in the receive tag/script:

```
<A HREF="##WT_SYSTEM.HREF#&Command=CONFIRM"> Continue </A>
```

```
<A HREF="##WT_SYSTEM.HREF#&Command=QUIT"> Cancel </A>
```

```
<wtOnReceiveScript>
<--
WT_HOST.comobj.Command.Value = WT_POSTED.Command;
!-->
</wtOnReceiveScript>
```

The HTML link is independent of the FORM tag and Dataform tag. Nevertheless, it can be entered within a Dataform tag.

4.4 Dialog between WebTransactions and the host application

Dialog control, i.e. the sequence of HTML pages in the browser corresponds to the format sequence at the host application in the case of an automatic 1:1 conversion (passive dialog). However, it is possible to modify the template and thus intervene actively in the dialog (active dialog).

You also have the option of verifying the sequence of the HTML pages issued and returned by the browser (synchronised dialog) or not (non-synchronised dialog).

4.4.1 Passive dialog

With the simplest form of dialog control, the host application is entirely responsible for determining the sequence of events. The resulting dialog sequence is the same as with terminal operation in which WebTransactions transfers the data for browser output. Each dialog cycle comprises the following actions:

- WebTransactions creates an HTML page from the current template. The host application form is converted to an HTML form.
- WebTransactions buffers the Receive tags/scripts for evaluation at a later stage.
- The HTTP server reads the HTML page and sends it to the browser.
- WebTransactions waits for data from the browser.
- The browser sends its data to the CGI program WTPublish.exe.
- WebTransactions the executes the Receive scripts. In the process, posted data (i.e. data sent by the browser) is transferred to the interface to the host application, and then forwarded to the host application. The next data record is received from the host application by setting the host objects and the send and receive methods.
- The next template depends on the form received. This is achieved using the receive method. From the system object attribute, WebTransactions determines the name of the form received from the host and specifies the corresponding template as the next template. The dialog cycle then begins again.

During a passive dialog, the steps described above are executed until the session is terminated in the template.This passive dialog control is already implemented in full and is ready for use when you configure WebTransactions for a particular host application (see section “Possible applications” on page 23). All automatic conversions can only implement a passive dialog. For active dialogs, you must intervene in the dialog sequence, i.e. in the templates.

4.4.2 Active dialog

Dialog control need not necessarily follow the format sequence of the host application. It is also possible to actively control the dialog using the template.

For instance, you can combine several host formats in an HTML page, thereby breaking the rule of one host format to one template. You then continue to communicate with the host application in a template until you have gathered all the data you wish to display on the HTML page.

An example of this chaining process is the option of scrolling up and down your screen (see the WebTransactions manual “Template Language” and the chapter “WTML tags”, section “DO UNTIL loop”). For instance, all the fields of a scrollable form are loaded before an HTML page is sent to the browser. Thus, at the end of one host dialog step, a follow-up step is automatically started without concluding the WebTransactions dialog cycle with browser output.

However, you can also actively determine the next template to be read by calling the embedded function `setNextPage()`. At the beginning of the next dialog cycle, WebTransactions uses the template specified.

This technique is used, for instance, if you wish to present a host format in several pages on the browser. It could also be used when communicating with several host applications if the end user wishes to interact with each host application.

A further example of a case where a dialog cycle need not involve contact with the host application or the browser is the option of selecting between different styles or languages. The data already received by the host in the previous step is presented again.

The interface offers buttons for switching between variants. Another template is thus evaluated for the HTML generation. This switching between templates is programmed by means of a Receive tag/script which assigns the appropriate value to the STYLE or LANGUAGE attribute of the system object. The FORMAT attribute in the system object remains unchanged. WebTransactions starts a new dialog cycle without contacting the host application.

In an extreme case, it is possible to program an application with WebTransactions without a host application. The dynamic data entered by the user on the HTML page could also be saved to a file, and interpreted as a catalog of orders for example.

4.5 Dialog between WebTransactions and the browser

When connecting host applications to the Web, the user interfaces of the host applications are used. From the host application's viewpoint, WebTransactions behaves in the same way as a terminal. It must therefore return forms received from the host application completed accordingly. To ensure that the logical states of the host application (e.g. open transactions of a database or transaction monitor) are not violated, the sequence of received and sent forms must be strictly observed.

In contrast, browsers are permitted to redisplay older HTML pages (History function) and resend the forms they contain. To avoid problems with the host application, WebTransactions can check whether the data received by the browser originated from the last HTML page created or whether an obsolete form was sent. WebTransactions then outputs an error message together with the last HTML page created. The dialog is thus resynchronized with the browser. This synchronized dialog is the most frequent form and is for instance used by all generated templates and all templates created by the Capture mechanism.

However, there are cases whether this rigid dialog represents an unnecessary restriction. For instance, it should be possible at any time to engage in a self-contained intermediate dialog with the host application or to perform a particular process without accessing the host application. To cater for this, WebTransactions also offers the option of the **non-synchronized dialog**, in which the page sent by the browser is not checked for relevance.

4.5.1 Synchronized dialog

In a synchronized dialog with the browser, WebTransactions ensures that only input from the last generated HTML page is processed. Processing takes place in dialog cycles which comprise the following steps:

- creation of an HTML page from a template
- user input at the browser
- processing of input (actions or statements in WTML tags or WTScripts of the template on Receive)

These dialog cycles are arranged in a defined sequence. You can view old output, for instance, and obtain information from the history (a feature that represents an extension to dialog control compared to the terminal). However, if you attempt to return one of these HTML pages, an error message is issued and the current page is output again.

There are two options for transferring user input from the browser to a WebTransactions session:

- as data of an HTML form, or
- in `QUERY_STRING` within the URL of a link.

In the case of synchronized dialog, use the `DataForm` tag without the `AsyncPage` parameter (see WebTransactions manual “Template Language”) or a link with the `HREF` attribute of the system object (see section “Starting by input of the URL” on page 94). The table below shows the result of the generation process:

Template	**HTML page**

`<wtDataForm>`	`<form method=“post“ action=...>`
	`<input type=“hidden“`
	`  name=“WT_SYSTEM_FORMAT_STATE“`
	`  value = ...> ...`

`<A`
`HREF="##WT_SYSTEM.HREF#">`
`...`
`</A>`
	`<A HREF=“/cgi-bin/WTPublish.exe`
	`?...&WT_SYSTEM_FORMAT_STATE=...">`
	`...`
	`</A>`

WebTransactions counts the dialog cycles with the browser within a WebTransactions session. The current value is provided in the `FORMAT_STATE` attribute of the system object or in a name/value pair within the `HREF` attribute. It is also generated as a hidden field in the HTML page during the conversion of a `DataForm` tag.

If WebTransactions is then addressed by such a form or link, a value for `WT_SYSTEM_FORMAT_STATE` is also returned. WebTransactions can compare this with the current counter for the dialog cycles and, if necessary, respond with an error message. If the values match, WebTransactions executes the OnReceive tags/scripts. The next dialog cycle then begins by reading the template specified in the `FORMAT` attribute.

4.5.2 Non-synchronized dialog

Open transactions and other limited dialogs prohibit free navigation in the dialog interface of a host application. However, situations often arise in which it is possible to abandon a rigid dialog without changing the logical state of the host application and without causing damage. For instance, you can include a button on an HTML page for displaying help information from the active host application in a separate window. This requires a self-contained intermediate dialog. Alternatively, you may wish to periodically ask the host adapter OSD whether asynchronous messages or print data has arrived on the active connection without having to send the current page from the browser each time. To do this, you can use a non-synchronized dialog which does not check that the interaction sequence is adhered to.

In the case of non-synchronized dialog, you use a link with the system object attribute HREF_ASYNC or the DataForm tag with the parameter AsyncPage.The following table shows the result of generation:

Template	**HTML page**
<A HREF="##WT_SYSTEM.HREF_ASYNC# &WT_ASYNC_PAGE=asPage"> ... </A>	<A HREF="/cgi-bin/WTPublish.exe ?...& WT_SYSTEM_FORMAT_STATE=IGNORE &WT_ASYNC_PAGE=asPage"> ... </A>
<A HREF="##WT_SYSTEM.HREF_ASYNC#"> ... </A>	<A HREF="/cgi-bin/WTPublish.exe ?...& WT_SYSTEM_FORMAT_STATE=IGNORE"> ... </A>

<wtDataForm asyncPage="asPage">	<form method=“post“ action=...>
	<input type=“hidden“
	  name=“WT_SYSTEM_FORMAT_STATE“
	  value=“IGNORE“> ...
	<input type=“hidden“
	  name=“WT_ASYNC_PAGE“
	  value=“asPage“> ...

The HREF_ASYNC attribute contains the name/value pair WT_SYSTEM_FORMAT_STATE=IGNORE. During the conversion of a DataForm tag with the asyncPage parameter, the value IGNORE is stored in the hidden field WT_SYSTEM_FORMAT_STATE, and the specification for asyncPage is converted to a hidden field WT_ASYNC_PAGE.

If WebTransactions is then addressed by such an HTML form or link, the value IGNORE transferred for WT_SYSTEM_FORMAT_STATE indicates that the sequence is not to be checked. WebTransactions is to respond to the request outside the rigid dialog. The template that is to process this request is determined by the value transferred for WT_ASYNC_PAGE.

This template should be designed such that it can be executed at any time. In principle, the template allows for communication with the host application, but this option should be exercised with great caution in view of synchronized access attempts. OnReceive scripts have no effect in these templates, as they require postprocessing within a synchronized dialog cycle.

The link with WT_SYSTEM_FORMAT_STATE=IGNORE with no entry for WT_ASYNC_PAGE represents a special case. In this case, the last synchronized page is output again. Example 2 shows how this function is used.

Example 1: Requesting help in a separate window

A host application offers an online help text for each input field. For this purpose, a question mark is inserted in the entry field and the forms are sent. This opens the first help page. Each help page comprises 23 output lines. The 24th line contains a command field for scrolling up and down (+ one page forward, - one page back) and for exiting the help system (`QUIT`). If further pages are available, the command field is preset to “+”; this changes to “-” as soon as you reach the end of the text. When you exit Help using the `QUIT` command, the host application returns to the original page.

Each HTML page is to be provided with a Help button. When this button is pressed, help information on the field in which the cursor is positioned is to be output in a separate window. The page itself is not to be confirmed.

To achieve this, enter the following lines in your template in an HTML form:

```
<INPUT TYPE="BUTTON" NAME="help" VALUE="Help"
onClick="window.open('##WT_SYSTEM.HREF_ASYNC#'+
'&WT_ASYNC_PAGE=HELP&CURSOR='+
document.forms[0].CURSOR.value,
'Help',
'scrollbars=yes toolbar=0 location=0'+
'status=0 height=400 width=400'+
'left=0 top=0 resizable=1')">
```

This generates a button which opens a new window displaying the appropriate help information. This is generated by means of non-synchronized access to WebTransactions. The `HELP` template creates the page (`WT_ASYNC_PAGE=HELP`). The template is informed of the name of the current field via the name/value pair `CURSOR=...` (it is assumed that the name of the current field is contained in `document.forms[0].CURSOR.value`). The template `Help.htm` is given below:

```
<HTML>
<HEAD><TITLE>Help</TITLE></HEAD>
<BODY>
<wtonCreateScript>
<!--

WT_HOST.STD[WT_POSTED.CURSOR].VALUE='?';                  Action 1
  first = true;                                           Action 2
  while ( first || WT_HOST.STD.E_24_001_04.VALUE != "-" ) Action 3
  {
  WT_HOST.STD.send();                                     Action 4
  WT_HOST.STD.receive();
  for (i=1;i<24;i++)                                      Action 5
```

```
   document.writeln( WT_HOST.STD['E_'+i+'_001_80'].HTMLVALUE,
'<br>');
  first = false;
 }

  for (i=1;i<24;i++)
 WT_HOST.STD.E_24_001_04.VALUE ="QUIT";              Action 6
 WT_HOST.STD.send();
 WT_HOST.STD.receive();
//-->
</wtonCreateScript>
<FORM>
<input type="BUTTON" name="CLOSE" value="Close"
onClick="window.close()“>                            Action 7
</FORM>
</BODY>
</HTML>
```

The template `HELP.htm` retrieves the help information from the host application, and navigates back to the original position. The synchronized dialog can then be resumed.

Details of the **actions** performed by `HELP.htm` are given below:

Firstly, the field containing the cursor in the HTML page is set to ‘`?`’ in the form (action 1). To ensure that the loop is run through at least once, `first` is set to `true` (action 2). Using the method calls `send/receive` (action 4), the first and all subsequent help pages are received in the loop (action 3). For each help page received, the first 23 lines of information are written to the output stream (action 5). The `QUIT` command returns you to the original page (action 6). A button is provided at the bottom of the page for closing the help window (action 7).

Example 2: Loading/unloading a frame during a session

Within a WebTransactions session, some HTML pages are to run in a frame. Buttons for operating the host application are to be displayed in a second control frame. However, it is **not** necessary for the frame set to be present for the entire session, rather only for certain HTML pages. When loading pages into the browser, therefore, they must be checked to establish whether or not they are to run in the dialog frame of the frame set. Some pages must load the frame set if they detect that it is not present; others must unload it if it is present. This is implemented as follows:

- Enter the following line at the beginning of all templates that are to run in the frame:

  ```
  <wtInclude name = frameOn>
  ```

- The template `frameOn.htm` contains the following:

  ```
  <SCRIPT>
  if( ! parent.ctr )
    self.location.href ="##WT_SYSTEM.HREF_ASYNC#&WT_ASYNC_PAGE=frameset";
  ```

```
</SCRIPT>
```

The client-side Javascript checks whether or not the frame set is already loaded (existence of the object `parent.ctr` for the control frame). If so, the page is displayed in the corresponding frame as normal. If the frame set is not present, it is loaded at the current position. This takes place by means of non-synchronized access to the WebTransactions session with the template `frameset.htm`.

- `frameset.htm` contains the following:

```
<HTML>
 <FRAMESET ROWS="20%,80%" BORDER=1>
  <FRAME NAME="ctr" SRC="##WT_SYSTEM.HREF_ASYNC#&WT_ASYNC_PAGE=control">
  <FRAME NAME="dlg" SRC="##WT_SYSTEM.HREF_ASYNC#">
 </FRAMESET>
</HTML>
```

The frame set contains two frames. In the first frame `ctr`, for instance, a tool bar is loaded by means of non-synchronized access to WebTransactions. However, this could also be defined as a static HTML page and loaded without WebTransactions.

The second frame also loaded by means of non-synchronized access to WebTransactions. In this case, however, no template is specified. WebTransactions therefore resends the last synchronized output. The current page is thus loaded into the frame `dlg`.

- Enter the following line at the beginning of all templates that are not to run in the frame:

```
<wtInclude name = frameOff>
```

- The template `frameOff.htm` contains the following:

```
<SCRIPT>
if( parent.ctr )
  parent.location.href = "##WT_SYSTEM.HREF_ASYNC#";
</SCRIPT>
```

The client-side Javascript checks whether or not the frame set is already loaded (existence of the object `parent.ctr` for the control frame). If not, the page is displayed as normal. If the frame set is present, the current page is loaded in its position (`parent.location`) and the frame set is unloaded. WebTransactions outputs the current page again by means of non-synchronized access without specifying `WT_ASYNC_PAGE`.

i Depending on the browser settings, this procedure may lead to problems when certain URLs are loaded from the browser cache. To force reloading from the server, you can release the cache or include an additional name/value pair with values modified step-by-step in the URL.

4.6 Terminating a session

A session can be terminated explicitly, by a timeout or via `WT_REMOTE`.

With WebLab it is possible to terminate any session started with WebLab explicitly.

4.6.1 Terminating a session explicitly

The user explicitly terminates a session in the browser. This means that you must give the user the option of terminating the session, for example with an Exit button as in the generated templates.

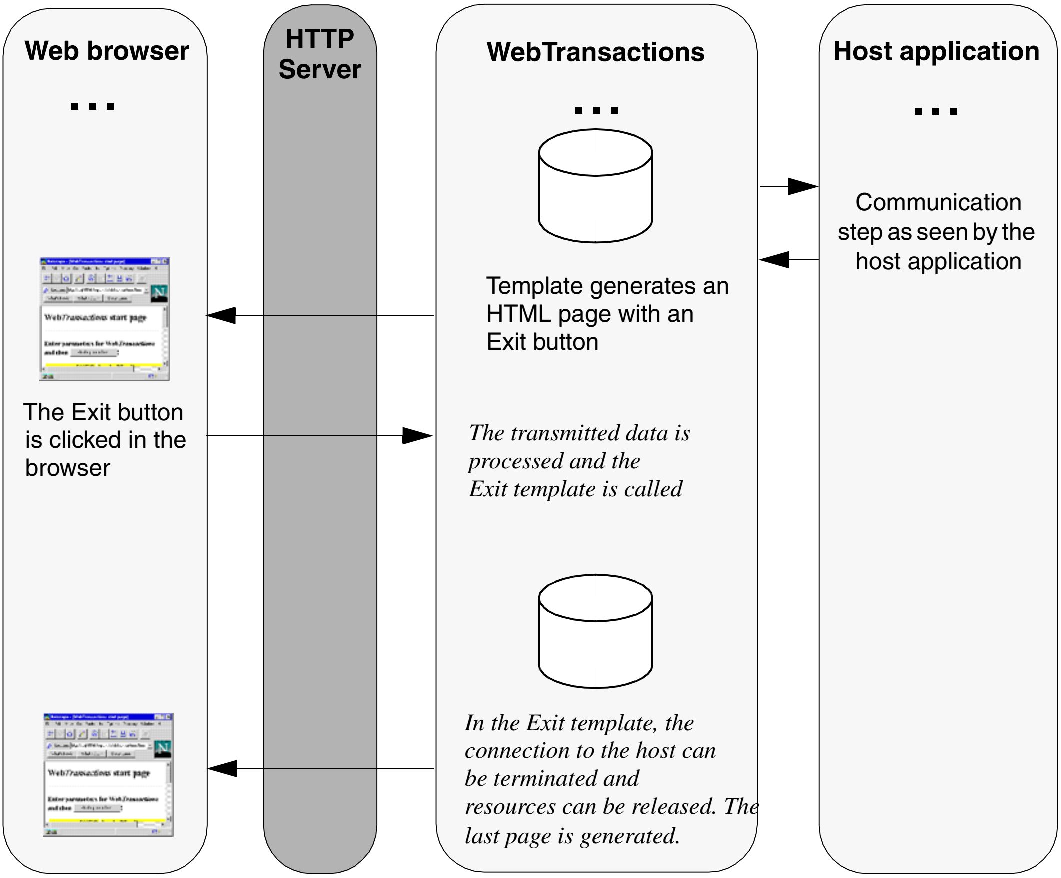

Web browser
HTTP Server
WebTransactions
Host application
...
...
...
Communication step as seen by the host application
Template generates an HTML page with an Exit button
The Exit button is clicked in the browser
The transmitted data is processed and the Exit template is called
In the Exit template, the connection to the host can be terminated and resources can be released. The last page is generated.

Figure 8: Explicit termination of a session

The last generated HTML page can no longer establish communications with the WebTransactions session, as this has already been terminated when the page appears in the browser. This means that `wtDataForm`-Tags or links with the URL `WT_SYSTEM.HREF[_ASYNC]` should not be used in the corresponding template. `onReceive` scripts in the last template are ignored and therefore serve no purpose.

Before termination, WebTransactions closes all open connections to host applications. If actions other than just closing are required for these connections, this must be explicitly programmed in the last template.

A session is terminated explicitly by calling the `exitSession()` function.

This statement creates the system attribute `EXIT_SESSION`.

If the `EXIT_SESSION` system attribute exists and the `PREVENT_EXIT_SESSION` system attribute does not exist, WebTransactions ends the session after outputting the next page, i.e.:

- If the action is performed when onCreate is executed, a page is created with the current template, sent to the browser, and the session is terminated.
- If the action is performed when onReceive is executed, a further template is read after the statement is processed, and a page is created and sent to the browser. Only then is the session terminated.

Example

```
HTML-Template
   ....
   <Input Type="SUBMIT" Name="Exit" Value="End">

   <wtRem Terminate communication with host>
   <wtonReceiveScript>
   if ( wt_Posted.Exit == "Exit")
   {
   WT_HOST.std.close();
   exitSession();
   setNextPage("final");
   }
   else
   {
   WT_HOST.std.send();
   WT_HOST.std.receive();
   }
   </wtonReceiveScript>
```

In this example, an option for immediate termination of the WebTransactions session is to be provided in generated templates.

To achieve this, you must integrate a corresponding input element in the page (in this case, a button labeled `Exit`). In an OnReceiveScript, a request will now be issued to establish whether or not this button has been activated. If not (`else` branch), the dialog with the host application is continued as normal (method calls `send()/receive()`). However, if you wish to terminate the session, the following steps are performed:

- the connection to the host application is closed (method call `close()`),
- termination of the WebTransactions session is requested (by calling the function `exitSession()`), and
- the system switches to the final template `final.htm`.

This template can contain a concluding text, and any links or forms required to start new WebTransactions sessions (see section “Starting a WebTransactions dialog application” on page 92). The template should not contain any `wtDataForm` tags or links with the URL `wt_System.HREF`, as these refer to a session that has already been terminated and will therefore result in an error message.

Predefined end template

With WebTransactions the general end template `wtend.htm` is supplied. `wtend.htm` can be entered for simple 01:01 connections, e.g. for the editing of `disconnect`, to obtain a completed connection.

When creating a base directory (see section “Creating a base directory” on page 160), a `wtend.htm` link is generated. If you edit and save `wtend.htm` in WebLab, the link is canceled and the modified file is saved in the base directory.

4.6.2 Terminating a session by means of a timeout

The HTTP connection between the Web server and the browser is shut down each time an HTML page is output. While a page is displayed at the browser, there is a logical connection but no physical connection between the browser and WebTransactions. If you exit the host application by shutting down the browser, for instance, the proxy process is not informed of this. To ensure that sessions exited in this way are not left waiting unnecessarily, WebTransactions supports a timeout mechanism.

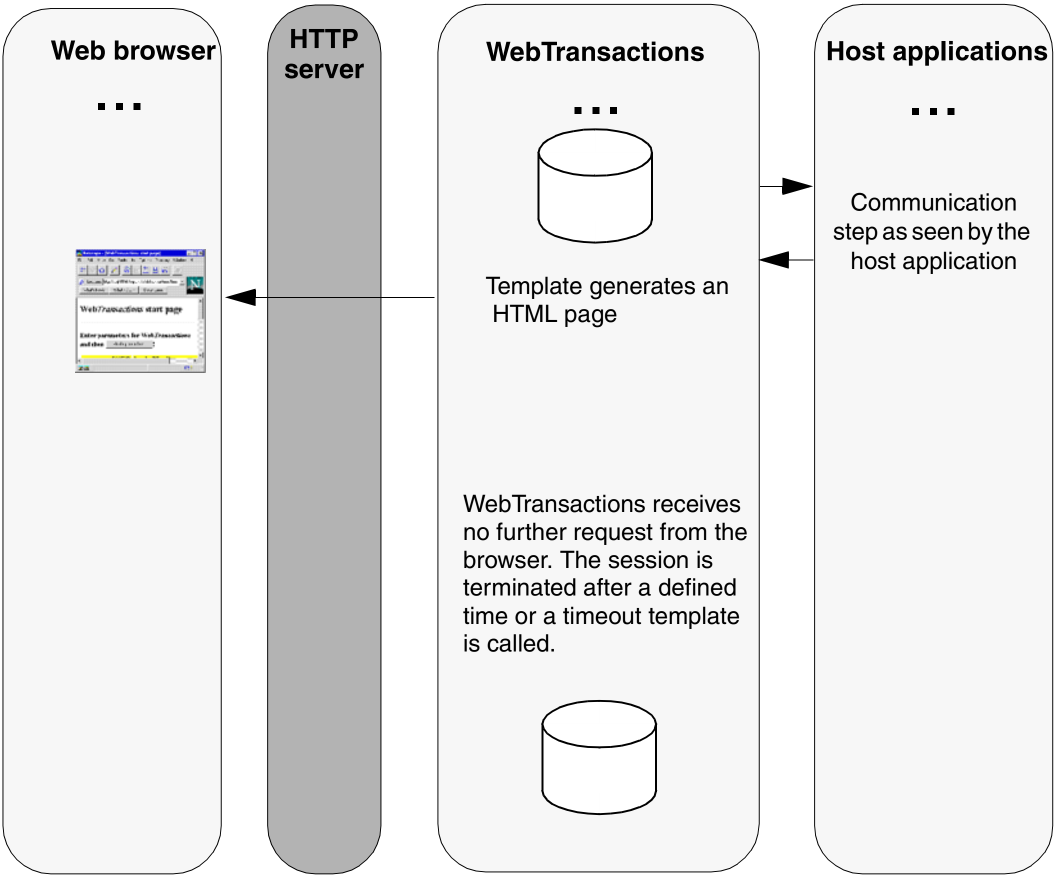

Web browser
...
HTTP server
WebTransactions
...
Host applications
...
Communication step as seen by the host application
Template generates an HTML page
WebTransactions receives no further request from the browser. The session is terminated after a defined time or a timeout template is called.

Figure 9: Terminating a session by means of a timeout

If no request is issued to WebTransactions within the timeout period, the session is terminated. Before termination, WebTransactions closes all open connections to host applications. If further actions other than closure are required for these connections, these must be programmed explicitly in a special timeout template. The name of this template is stored in the TIMEOUT_FORMAT attribute of the system object. WebTransactions executes this template as its final action before terminating the session.

i You should note that a timeout template must not contain any output to the browser since it is only called when the connection to the browser no longer exists.

The number of seconds to be spent by WebTransactions waiting for the next request from the browser is specified in the TIMEOUT_USER attribute of the system object. For example, the following statement sets a timeout period of 5 minutes:

```
<wtOn..Script>
  WT_SYSTEM.TIMEOUT_USER = "300";
</wtOn..Script>
```

If the TIMEOUT_USER attribute is not set explicitly in a template, WebTransactions assumes a timeout period of 10 minutes.

Example

In this TIMEOUT template not only an existing connection is terminated, but also the user is logged off with a special command before termination.

```
<wtOnCreateScript>
<!--
   WT_HOST.osd.send();
   WT_HOST.osd.receive();
   WT_HOST.osd.E_001_001.VALUE="LOGOFF SYSTEM_OUTPUT=*TAPE_OUTPUT";
   WT_HOST.osd.send();
   WT_HOST.osd.receive();
   WT_HOST.osd.close();
//-->
</wtOnCreateScript>
```

4.6.3 Terminating via WT_REMOTE

A client program terminates a session via the interface WT_REMOTE using the method EXIT_SESSION.

4.7 Diagnoses in a WebTransactions application

A variety of trace functions can be used for diagnostic purposes in a WebTransactions application. It is also possible to record sessions.

Single step tracking is another method of diagnosis available in WebLab, see also section "Tracking execution via a template or a template area" on page 193.

4.7.1 Trace functions

You can set different traces for diagnostic purposes:

- communication traces for WebLab and WTEdit
- a session trace for WebTransactions

4.7.1.1 Communication traces

To permit the rapid identification of communication problems between WebLab and WTEdit you can activate two traces in WebLab using **Options/Preferences/Diagnosis**:

- the WebLab trace
- the WTEdit race

Both traces log communication between WebLab and WTEdit:

- the WebLab trace logs WebLab communication locally in the specified file
- the WTEdit trace logs WTEdit communication at the WebTransactions server in the specified file

Both files are required for rapid diagnosis. They provide a complete overview of communications.

4.7.1.2 WebTransactions trace

The WebTransactions trace logs the entire session. The trace file contains information from the following sources:

- internal function calls
- host adapter function calls
- user exit calls (provided that these write trace entries)
- the global WTScript function `writeToTrace()` (for any user-defined trace entries)

Activating tracing in WebLab

In WebLab there are two ways of activating or deactivating the WebTransactions trace:

- by choosing the commands **Options/Activate Trace** or **Deactivate Trace**
- by double-clicking on the T icon in the status bar

> i This setting remains active even after the session has been terminated. You must explicitly deactivate the trace.

If trace mode is active, WebTransactions creates a file in the base directory under `tmp/`*session_id*`/Trace`. As long as the session is active, you can find out the session ID with **Option/Show Information**.

You can view the trace file with the commands:

- **File/Open**
- **Administration/Application**

Activating tracing for an application

You activate the trace function for a WebTransactions application by starting WebTransactions via an HTML start page which contains a Form tag for the method POST. The form must contain an input element with the name `WT_TRACE` for which the value `Trace` is set. This can be a hidden input field as presented below. You can also use input fields of a different type and, for example, display a checkbox to make it possible to select whether or not the trace function is to be activated.

```
<form method="post" action= "/cgi-prefix/WTPublish.exe/basedir?startTemplate">
<input type="hidden" name="WT_TRACE" value="Trace">
```

The `WT_TRACE` field must have the value `Trace` before you can activate the trace function. All other values (including undefined) deactivate the trace.

Activating tracing in a template

Instead of using `WT_TRACE`, you can also use the global function `setTraceLevel()`. You can use `setTraceLevel("FULL")` to activate the trace during an open session and then use `setTraceLevel("NONE")` to deactivate it again.

In this way, the trace can be restricted to certain parts of the template in order to keep the trace file small and restrict it to defined sections of the session. The function is described in the WebTransactions manual “Template Language”.

Defining the trace level for host adapters

For the OSD and MVS host adapters, you can not only activate or deactivate the trace but also define the level at which tracing is to be performed. You can use the connection-specific system object attribute `TRACE_LEVEL` to define what is to be logged:

– various trace levels from 0 to 3

– `E:` output of emulation function calls

– `M:` output of all host matrixes, i.e. the “raw” screen data

For more detailed information, see the WebTransactions manuals for the corresponding host adapters.

4.7.2 Recording a session

At runtime, communications between the WebTransactions kernel and the host application are performed via the terminal emulation that is integrated in the host adapter. By recording the session, you log all the messages to and from the host application. You can “play back” an emulation trace in another session. I.e. the recorded session is run again even without a connection to the host (“offline”). In this way, you can continue to develop a WebTransactions application without being connected to the host.

To record a session, set the system object attribute `RECORD_HOST_COMMUNICATION` in the start template to “Yes”. By default, the file containing the recorded emulation trace has the name
`WEBTADUMP.LOG`. However, you can define a different name by means of the system object attribute `OFFLINE_LOGFILE`.

```
RECORD_HOST_COMMUNICATION="Yes"
OFFLINE_LOGFILE=<filename>
```

“Playing back” the recording

You can play back a recording without being connected to the host application by setting the following system object attributes:

```
OFFLINE_COMMUNICATION="Yes"
OFFLINE_TRACEFILE="<trace filename>"
```

Since there are different system object attributes for the files in which you record and play back a session, you can create a new recording and play back an existing one in one and the same session.

4.8 Transferring a WebTransactions application

WebTransactions applications are usually developed and tested at a development machine. They are then transferred to a production computer for productive use. Maintenance and further development continue to be performed on the development machine and the version installed on the production computer must be updated from time to time.

A wizard is available to assist you when transferring a WebTransactions application from one machine to another. To do this, WebLab is used to pack all the files and subdirectories below a base directory into an archive which you can transfer to another computer and then unpack with WebLab. As a result, the base directory on the production computer is set up in exactly the same way as on the development machine.

You can distribute WebTransactions applications to cluster members using the **Administration/Distribute Application** command.

4.8.1 Unpacking an application by command

If you cannot access a WebTransactions server with WebLab then you can use the `install` command provided by `WTEdit.exe` to unpack a WebTransactions application. This command creates a base directory on the server at which it is executed and unpacks the specified WebTransactions application in this base directory. You call the command as follows:

`wtedit.exe install` *name basedir id password wtpublish-path isapi-path cluster-path*

i You must specify all the parameters.

name	Name of the archive to be unpacked together with its absolute path
basedir	Base directory that is to be generated together with its absolute path
id	WebTransactions user identification
password	Password corresponding to the user identification
wtpublish-path	Alias name for the path to WTPublish.exe
isapi-path	Alias name for the path to WTPublishISAPI.exe
cluster-path	Alias name for the path to WTCluster.exe

Example

This example unpacks the file *packtest.zip* to the base directory *osd_test*. The user ID is *ac13* and has no associated password. The command must be entered without line breaks.

```
wtedit.exe install "d:\projects\packtest.zip" "d:\basedirs\osd_test" ac13 ""
cgi-bin cgi-bin cgi-bin
```

4.9 Client interface WT_REMOTE

WT_REMOTE provides an interface for any client programs and allows WebTransactions applications to be used for these client programs. The interface allows access not only to the data of the underlying dialog applications, but to all the resources of the WebTransactions session.

WT_REMOTE allows a current WebTransactions session to be accessed. This is done by specifying the session ID when the client accesses WebTransactions. A WebTransactions session can also be started independently by a client program. This allows both single-step transactions and multi-step transactions:

– With a single-step transaction, a WebTransactions session is started in order to execute a single command. The command is then executed and the session is terminated. This is done with a single client access to WT_REMOTE.

– With a multi-step transaction, a WebTransactions session is started explicitly with the WT_REMOTE access START_SESSION. Then a number of client accesses are carried out with PROCESS_COMMANDS and finally the session is terminated with EXIT_SESSION.

WT_REMOTE provides the methods START_SESSION, PROCESS_COMMANDS and EXIT_SESSION for communication with the WebTransactions application. These methods enable clint access. START_SESSION and EXIT_SESSION are the methods used for explicitly starting and terminating an application within a multi-step transaction. PROCESS_COMMANDS is used to execute the actual client accesses (for both single-step and multi-step transactions).

The PROCESS_COMMANDS method allows data to be transferred from the client program to the WebTransactions application and vice versa. It also allows objects to be created in the WebTransactions session and methods to be called in the WebTransactions session.

There are many ways of using the WT_REMOTE interface:

– WebTransactions sessions can use the WT_REMOTE interface via the supplied libraries in order to communicate with each other, thus allowing distributed WebTransactions sessions (WT.RPC).

– Using predefined classes, JAVA applets can access WebTransactions sessions and use the services provided for their own purposes.

– You can also combine dialog applications and client programs. For example, an HTML page can be generated with an applet in a WebTransactions dialog application, and the applet can in turn communicate with the WebTransactions application via WT_REMOTE.

– On the basis of the open WT_REMOTE interface, you can implement your own client programs for WebTransactions.

For a detailed description of the client interface, see the WebTransactions manual “Client APIs for WebTransactions”.

4.10 Administering a WebTransactions application

You can administer your WebTransactions application in full from the browser. To help you do this, WebTransactions provides you with an administration program which is based on our own security and user concepts. For more information, see section “User concept” on page 132.

The administration program comprises the following **functions**:

- displaying sessions
- locking and unlocking WebTransactions; when locked, no more sessions are set up
- terminating current sessions
- displaying temporary files relating to current sessions
- cleaning up the `tmp` directory

Calling the administration program

You can start the administration interface either in WebLab or directly in the browser via a separate URL.

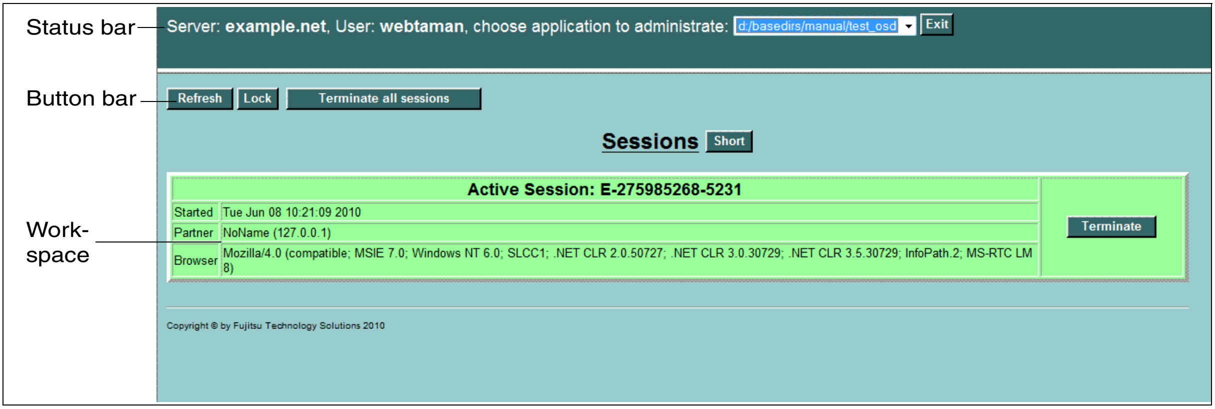
Call in WebLab*:*
Select **Administration/Application**.

Call via URL:
Start any browser and then start the administration program using the following URL:
`http://`*webta-server/cgi-prefix*`/WTPublish.exe/server-admin`,
where *webta-server* is the name of the host on which WebTransactions is running and *cgi-prefix* is the path to this host for CGI programs.

The administration program is started and the logon window is displayed in the browser. Log on with the ID which you have been assigned by the WebTransactions administrator in order to administer your own WebTransactions application. You will now see the window **WebTransactions Application Administration**.

Structure of the administration interface

The Figure below presents the administration interface:



Server: **example.net**, User: **webtaman**, choose application to administrate:

Sessions

Active Session: E-275985268-5231

Started	Tue Jun 08 10:21:09 2010
Partner	NoName (127.0.0.1)
Browser	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; InfoPath.2; MS-RTC LM 8)

Copyright © by Fujitsu Technology Solutions 2010

The user interface of the administration program for application management consists of three areas:

- The status bar which displays the name of the current WebTransactions server and the user. If you are editing more than one WebTransactions application on the WebTransactions server, you can select the one you want to administer from a list.
- A button bar with which you can control administration. Depending on the situation on the WebTransactions server, the following buttons may be displayed:

  Refresh
  Updates the contents of the page.
  (The page is also automatically updated at regular intervals. The bar at the top edge of the status bar indicates the time until the next automatic refresh: the shorter this bar is, the shorter the time.)

  Lock
  Indicates that WebTransactions will permit new sessions. If you click on this button, the label changes to **UnLock** and the WebTransactions application switches to locked mode. This means that current sessions remain active but no further sessions can be started. To unlock the WebTransactions application again, click on the **Unlock** button.

  Terminate all Sessions
  Terminates all current sessions.

  Delete all files of dead sessions
  Deletes all files belonging to terminated sessions. Current sessions are not affected.

Remove all files not related to any session
Deletes all the files in the temporary directory which cannot be assigned to a session. If present, such files are displayed in a list.

- The workspace in which all the sessions together with their resources and session IDs are displayed. The following entries can be displayed in the workspace:

Dead Session

If files (e.g. trace files) are still present in the session directory when a session is terminated, the session continues to be displayed as a “dead session” and it is still possible to access the session files (via the corresponding link in the line `Session Files`). You can use the **Delete** button to delete these files.

Active Session

In the case of active sessions, the administration interface displays the following information:

– the operating system resource for process communication between WebTransactions and the browser or host application (named pipes)

– the start time of the session

– the Internet address of the computer on which the browser is running (or possibly the address of a proxy computer)

– information about the browser

– temporary session files (e.g. print files) and the trace file

You can use the **Terminate** button to end any of these sessions, i.e. terminate the holder task and delete any associated temporary files.

If the user of this deleted session then sends the current HTML page from the browser, WebTransactions issues an error message. The user must then open a new session.

If a trace file exists for the displayed session, the name of this file is output under **Session Files**. You can use the corresponding link to view temporary files.

5 WebTransactions server

You can completely administer the WebTransactions server from the browser. For this, WebTransactions provides an administration program, which is based on its own security and user concept, see section “User concept” on page 132. The administration program is a standalone WebTransactions application, which can be called using WebLab or directly from the browser. The administration program itself is always executed on the WebTransactions server.

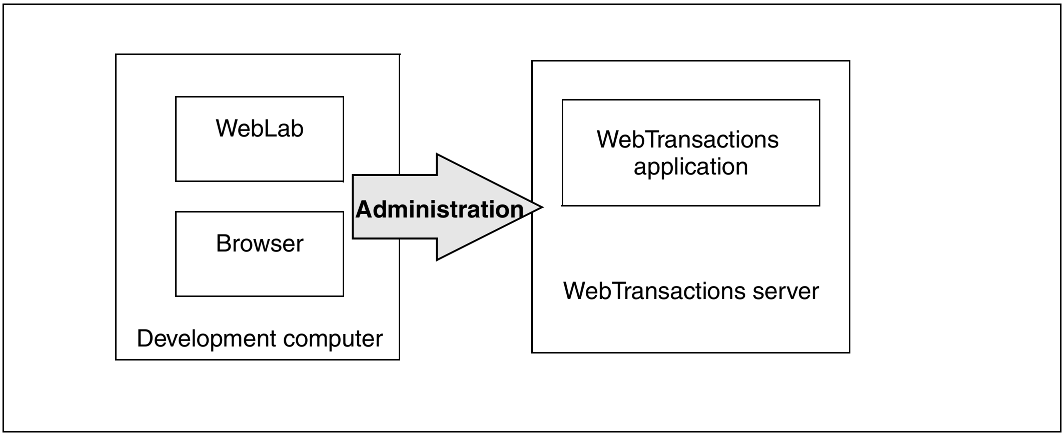


Figure 10: Administration options

The administration program contains the following **functions**:

- Enter or upgrade licenses (see page 135)
- Administer the WebTransactions server, i.e. basic administration of users, pools, applications, sessions, tools, clusters (see page 139)
- Set up and manage clusters (see page 145)

5.1 User concept

The administration of WebTransactions is based on a special user concept which is implemented on the basis of the access controls on the WebTransactions server and on WebTransactions applications.

WebTransactions recognizes two types of user: `admin` and other `user`s.

`admin` corresponds to the administrator under Windows or the superuser in Unix systems, and is authorized to do everything in the WebTransactions environment on the WebTransactions server. The user `admin` is the WebTransactions server administrator and is set up without a password during the installation of WebTransactions. This user holds all privileges and cannot be deleted.

i Remember to give the user `admin` a secure password as soon as installation is completed. You can change this password at any time.

The other `user`s are set up by the `admin` user and have those rights and access options granted to them by the `admin` user. A `user` can, for example, take on a role which corresponds to their tasks:

- The application administrators may manage WebTransactions applications. They can monitor sessions, evaluate trace files, lock or release WebTransactions applications, or terminate sessions.
- The application developers may use WebLab to create new WebTransactions applications and develop existing applications.

Thus, the individual application programmers do not need to worry about the settings of the operating system or the Web server and can assume that their WebTransactions applications on the server are protected from unauthorized access.

The management functions are dependent on the role fulfilled by the user:

- The WebTransactions administrator can manage the WebTransactions server as an individual server and as part of a cluster. This form of administration of a WebTransactions server is described in the following sections.
- Each application programmer can manage their WebTransactions application in the same way as before using the new administration program. In WebLab, you use the command **Administration/Application**, to call the administration program.This form of administration is described in the relevant product-specific manuals.

5.2 Starting the administration program

You start the administration program either using WebLab or directly in the browser via a special URL.

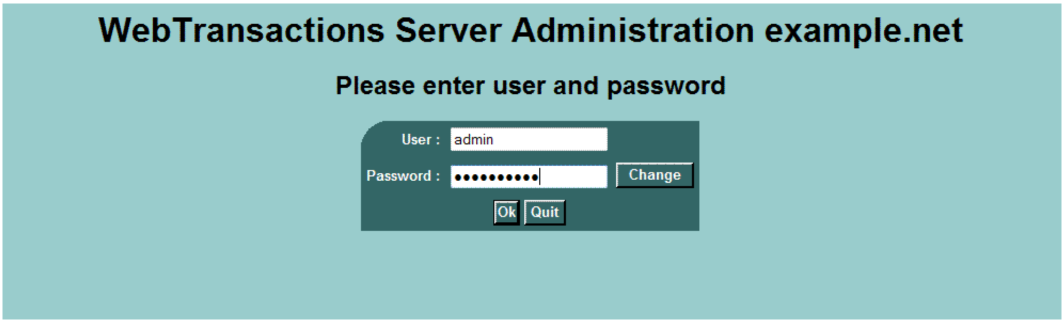
Call in WebLab*:*

Select the command **Administration/Server**.

Call via URL:

Start the browser of your choice, and start the administration program with the following URL: `http://`*webta-server/cgi-prefix*`/WTPublish.exe/server-admin`. *webta-server* is the name of the computer on which WebTransactions, *cgi-prefix* is the path on this computer for CGI programs.

This starts the administration program and displays the login window in the browser.



WebTransactions Server Administration example.net

Please enter user and password

User : admin

Password : ••••••••••

When you initially start the WebTransactions administration program, you should first set up a password for the user `admin`. Clicking on the **Change** button allows you to either set a password or change an existing one.

Every other time you start the administration program, the view will depend on the ID under which you log in:

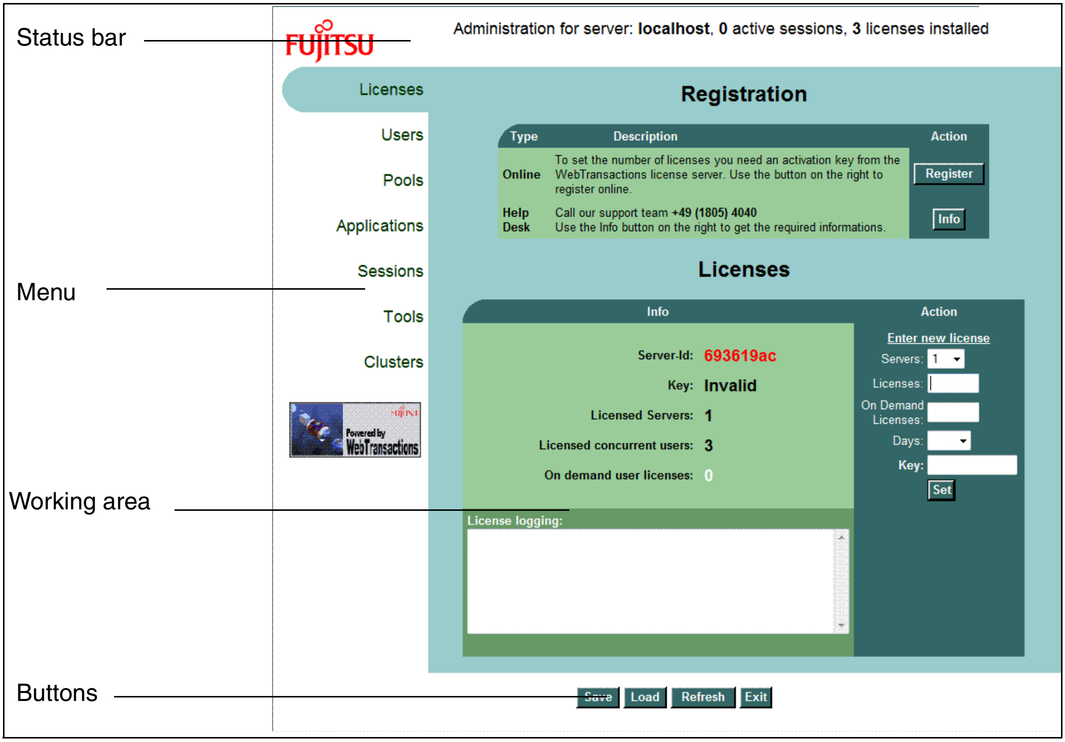
`admin`

You are the server administrator of WebTransactions

other `user`

You have the privileges granted by `admin`, see also section “Administering a WebTransactions application” on page 127.

After you have logged in as `admin`, the first window of the administration program will be displayed in the browser:



Administration for server: **localhost**, **0** active sessions, **3** licenses installed

Registration

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

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: 1

Licenses:

On Demand Licenses:

Days:

Key:

Set

The interface of the administration program consists of

- A status bar containing information about the server to be managed, the number of sessions and licenses
- A vertical menu on the left of the window contains the menu items **Licenses**, **Users**, **Pools**, **Applications**, **Sessions**, **Tools** and **Clusters**. The actual appearance of the menu will depend on the browser you are using and the supply unit installed.
- A working area to the right of the menu in which you carry out the actual work.
- Buttons which allow you to save or load your settings, refresh the contents of the browser window or terminate the administration program. Any changes you make during administration only become effective when you click on the **Save** button.

i All settings that you enter for the administration of the WebTransactions server and applications are saved in ASCII format in the file called `config/wtaccess` in the installation directory of WebTransactions.

5.3 Entering or upgrading licenses

WebTransactions offers the following license models:

– Standalone licenses

– On-demand licenses

– Cluster licenses

The following sections describe the detailed procedure for entering licenses.

The license page is displayed automatically when you log on to the administration program.

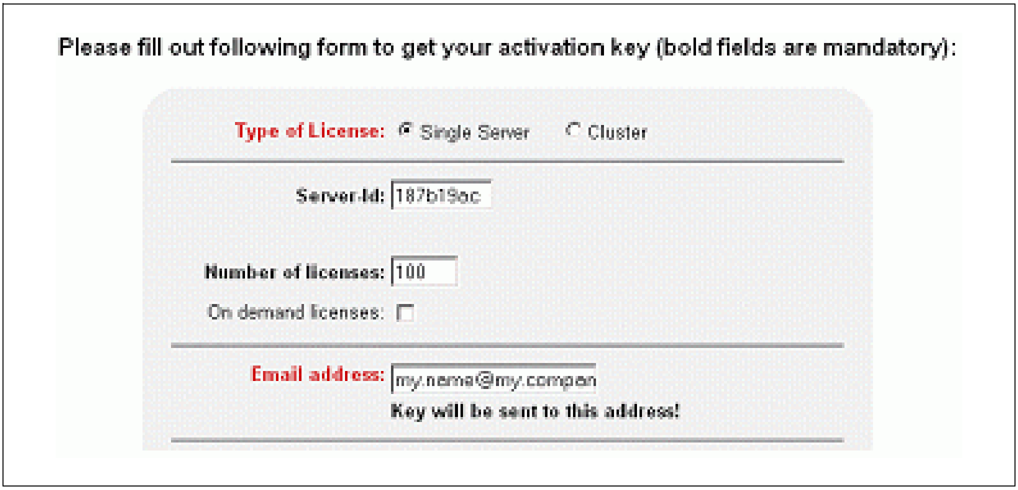
5.3.1 Standalone licenses

A server on which WebTransactions is installed is licensed for a maximum number of users simultaneously.

Proceed as follows to enter new standalone licenses or upgrade existing licenses:

► Click the **Register** or **Change Registration** button on the license page.

The registration page opens



Please fill out following form to get your activation key (bold fields are mandatory):

Type of License: ● Single Server ○ Cluster

Server-Id: 187b19ac

Number of licenses: 100

On demand licenses: □

Email address: my.name@my.compan

Key will be sent to this address!

► To register licenses for a standalone server, select the option **Single Server** under **Type of license**.

► Enter the number of licenses purchased under **Number of licences**.

► Enter your e-mail address and other parameters if required.

- Send the form with **Request Key**.

  The license key will soon be sent to the specified e-mail address.

- Enter the number of licenses purchased and/or the valid license key sent to you by e-mail in the **Licenses** and **Key** fields in the license page and confirm by selecting **Set** followed by **Save**.

  The licenses are activated. The new number of licenses is displayed in the status bar.

5.3.2 On-demand licenses

The number of users who are needed at any one time often reaches a temporary peak value. Additional, economical on-demand licenses are available to help you respond to such periods of peak load.

You can use on-demand licenses for both standalone servers and for cluster licenses.

Example

“We need 2000 extra users, but only on Mondays. Otherwise 500 is enough.”

In the past, this load profile demanded the licensing of 2000 simultaneous users if reliable operation was to be guaranteed.

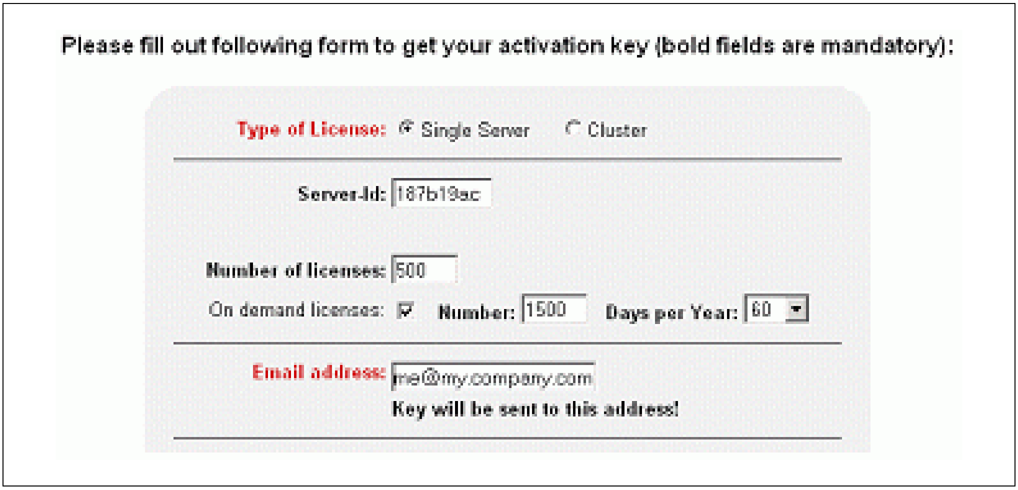
With the introduction of the new extended license model, licenses are required for only 500 simultaneous users together with 1500 less expensive on-demand licenses for 60 days of the year. 500 users can now work simultaneously at any time. As soon as the 501st session is started, one day is deducted from the “on-demand days” and 2000 sessions can run in parallel for 24 hours.

At the end of a year, the licensed number of days is again available for the next year without it being necessary to re-license. The year starts at the date of registration.

Proceed as follows to enter new on-demand licenses or upgrade existing ones:

- ► In the license page, click the **Register** or **Change Registration** button.

  The registration page is opened



Please fill out following form to get your activation key (bold fields are mandatory):

Type of License: ● Single Server ○ Cluster

Server-Id: 187b19ac

Number of licenses: 500

On demand licenses: ☑ Number: 1500 Days per Year: 60

Email address: me@my.company.com

Key will be sent to this address!

- ► In the registration page, specify whether you require on-demand licenses for a standalone server (**Single Server** option) or for a cluster (**Cluster** option).
- ► Select the option **On demand licences**.

  The **Number** input field for the number of on-demand lists and a **Days per Year** selection list for the number of on-demand days (30, 60, 90 or 120) are now displayed.
- ► Specify the number of on-demand licenses and the number of on-demand days.
- ► Enter your e-mail address and additional parameters if necessary.
- ► Send the form with **Request Key**.

  The license key will soon be sent to the specified e-mail address.

► Enter the following information in the licensing page:

Field	**Specification**
Servers	Number of servers for which you need licenses
Licences	Number of licenses purchased
On Demand Licences	Number of on-demand licenses
Days	Number of on-demand days
Key	Valid license key that was sent to you by e-mail

► Confirm with **Set**, followed by **Save**.

The licenses are activated. The new number of licenses is displayed in the status bar.

In the license page

– you can check the current “used days” status at any time in the **Info** area,

– the date of all used on-demand licenses is logged in the **License Logging** field.

5.3.3 Cluster licenses

To permit the convenient, flexible set-up of WebTransactions clusters on modern hardware, for example on blade servers, it is necessary to be able to perform license administration independently of the servers that are currently in use. For this reason, there are cluster licenses that only need to be entered on the cluster controller and apply to all the servers in the cluster.

For detailed information on entering and upgrading cluster licenses, see section “Registering cluster licenses” on page 143.

5.4 Managing the WebTransactions server

When you call the administration program for the first time you should first enter the number of licenses you have. You should only have to do this once. Should you discover you do not have enough licenses, you can purchase more and then, in the administration program, adjust the number of licenses accordingly.

For information on registering licenses, see section "Entering or upgrading licenses" on page 135.

The administration program has an intuitive interface. During the development phase the administrator will be mainly responsible for users and directories. Each application developer can manage their own WebTransactions applications in accordance with the settings made in the administration program. This privilege must be actively revoked by the administrator.

Basic operation of the administration program

The following basic processing options are available for each menu item in the administration program, e.g. **Users**, **Pools**:

For details that go beyond the scope of this basic introduction to operation, see the example sessions for the various host adapter variants and the detailed WebTransactions online help system.

Create, e.g. create user

- ► Click on the required menu item in the drop-down menu.

  The window corresponding to the menu item is opened with a list of existing components, such as users or pools, together with input fields for new components.

- ► Enter the required new components in the corresponding input fields.
- ► Click **Add** to add the newly created component to the list.

Edit properties, e.g. edit a user's properties

- ► Click on the required menu item in the drop-down menu.

  The window corresponding to the menu item is opened with a list of existing components.

- ► Click on the existing component that you want to edit.

  The properties of this component are displayed.

- ► Select a required or superfluous component and click **Add** or **Delete**.

or

- ► Activate or deactivate options to assign or withdraw the corresponding properties.

  Whenever you click on an option (checkbox), the form is sent immediately.

Delete an existing component, e.g. delete a user

- ► Click **Remove** to delete an existing component.

Scope

The basic operation of the administration program (see above) applies, in particular to the following functions:

- Creating and editing users

  Only the users you have set up using the administration program are permitted to administer WebTransactions applications or work with WebLab.

- Setting up pools and editing properties

  A WebTransactions pool is a directory in which you can create base directories using WebLab. You can only set up base directories with WebLab in pools that you have set up and declared using the administration program.

- Editing the properties of WebTransactions applications

  WebTransactions applications are generally set up on the server using the WebLab command **Project/New...**. The user who creates the application receives administration and edit privileges for this application.

  You call application administration directly with the **Administrate** button.

- Sessions

  The distribution of the open sessions to the local base directories is displayed. Only those base directories in which there are open sessions are displayed.

  Click the **Administrate** button to call application administration directly. You can then click on the base directory to edit the application's properties. If the button or link is missing then the base directory has not been set up for administration via the GUI. This is the case, for example, of the administration program itself.

- Defining tools and editing their properties
- Setting up clusters and editing their properties

  ➪ See also section "Setting up a cluster" on page 145 and section "Editing a cluster's properties" on page 146.

5.5 Cluster concept

Depending on the performance of the processor and the main memory configuration, the WebTransactions server has a maximum number of parallel WebTransactions sessions which, if exceeded, causes performance to drop to level which is no longer acceptable. If you wish to enable further parallel sessions on a WebTransactions application, you must distribute the sessions across a number of servers, This can easily be achieved using a WebTransactions cluster.

Several integration servers can be combined to form a so-called cluster. The integration servers do not necessarily need to be of the same type. All integration platforms supported by WebTransactions, Windows, OSD and Unix platforms can be combined in any way you choose. For this to be possible, WebTransactions must be installed on all the computers of the planned cluster and the relevant host adapters for the appropriate WebTransactions platforms must also be available. The cluster is defined and set up using the administration program.

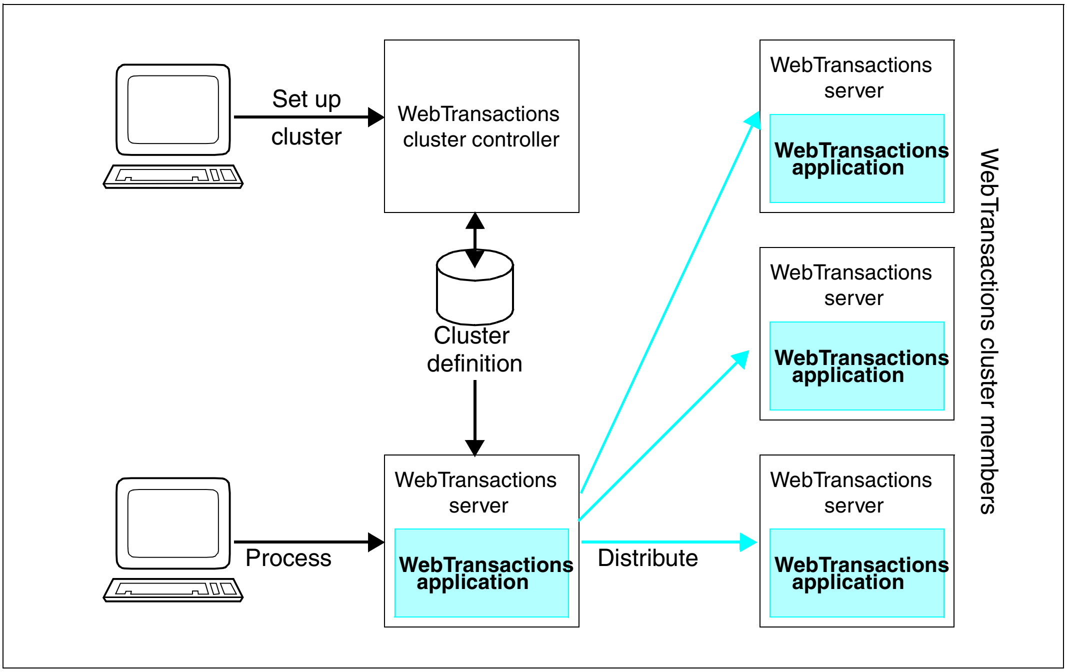

Set up
cluster
WebTransactions cluster controller
Cluster definition
Process
WebTransactions server
WebTransactions application
Distribute
WebTransactions server
WebTransactions application
WebTransactions server
WebTransactions application
WebTransactions server
WebTransactions application
WebTransactions cluster members

Figure 11: Setting up a cluster

One or more of the servers in the cluster take on the role of the cluster controller. This server stores the cluster definition. The other integration servers, also known as cluster members, each have a copy of the master application. The master application is the one distributed across the cluster members. Other than this, it is no different from a “normal” WebTransactions application.

There is a wizard in the WebLab development environment which can be used to help distribute the master application across the cluster members (**Administration/Distribute Application** command). This reads the cluster definition and then copies the base directory of the master application to the cluster member. You can use cluster licenses for joint license administration, see also section “Registering cluster licenses” on page 143.

Procedure

- ► First, use WebLab to create a master application on the WebTransactions server.
- ► Then, use the administration program to set up a WebTransactions cluster on the cluster controller. The cluster controller is that computer on which the cluster definition has been stored. The URL at the start of a session on the cluster also points to this server. The cluster controller then decides on which cluster member the WebTransactions session is to be started.
- ► Use WebLab to distribute the master application to the cluster members using the command **Administration/Distribute Application**.

5.5.1 Registering cluster licenses

Cluster licenses are valid for all the servers involved in WebTransactions clusters if these are addressed via the licensed server that acts as cluster controller. Although WebTransactions must be installed and applications set up on these computers, they no longer need to be registered individually.

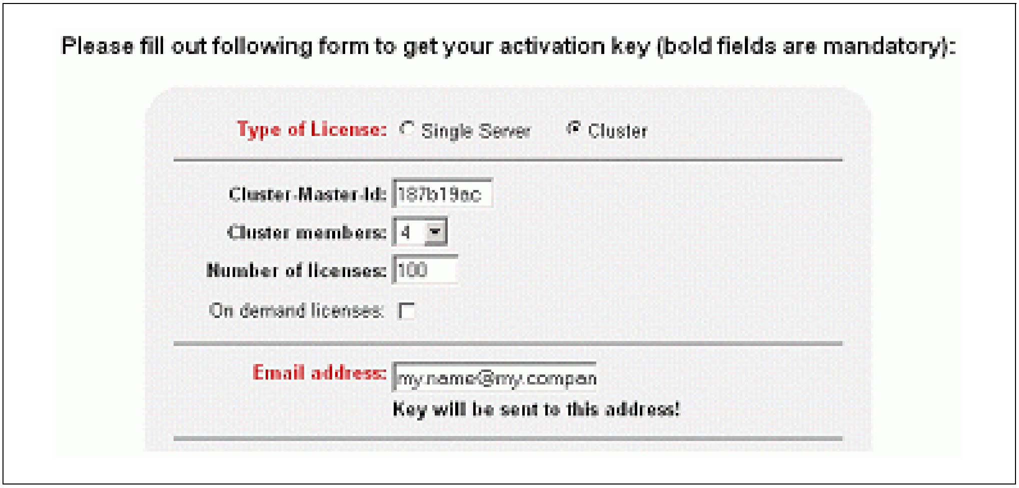
The number of registered servers is the maximum number of computers that may take part in the WebTransactions cluster. Individual clusters may also contain fewer individual computers and cluster members may occur in multiple cluster definitions.

A cluster may also include computers that are not licensed via the cluster but via individually entered server licenses. WebTransactions automatically determines the license type to be used when distributing the sessions.

Proceed as follows to enter new cluster licenses or upgrade existing ones:

- Click the **Register** or **Change Registration** button on the license page

  The registration page is opened



Please fill out following form to get your activation key (bold fields are mandatory):

Type of License: ○ Single Server ● Cluster

Cluster-Master-Id: 187b19ac

Cluster members: 4

Number of licenses: 100

On demand licenses: □

Email address: myname@my.compan

Key will be sent to this address!

- Select the **Cluster** option under **Type of license** in the registration page.

  The **Cluster members** selection list is now displayed.

- In the **Cluster members** selection list, select the number of servers that you want to license.

- Enter the number of licenses purchased in the **Number of licences** field.

- Enter your e-mail address and additional parameters if necessary.

- Send the form with **Request Key**.

  The license key will soon be sent to the specified e-mail address.

- Enter the following information in the license page:

Field	Specification
Servers	Number of cluster members
Licences	Number of licenses purchased
Key	Valid license key that was sent to you by e-mail

- Confirm your specifications with **Set**, followed by **Save**.

  The cluster is licensed.

The cluster controller is entered automatically and cannot be deleted.
Proceed as follows to enter the servers that are to use this cluster license:

- Click on **Clusters** in the drop-down menu.

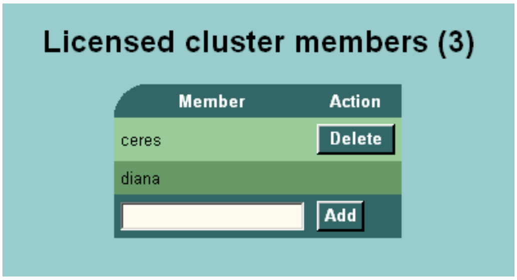
  The **Licensed cluster members** list is displayed.

- To enter a new server, enter the server name in the text box and click **Add**.

  This function is available until the maximum number of cluster members is exhausted.

- To delete an individual server, click **Delete.**

- To activate the modified configuration, save it with **Save**



Licensed cluster members (3)

Member	Action
ceres	Delete
diana	
	Add

5.5.2 Setting up a cluster

- For information on setting up a cluster, see section “Basic operation of the administration program” on page 139.
- For details that go beyond this introduction to basic operation, see the detailed WebTransactions online help system.
- The administration facility that is displayed when you click the **Administrate** button corresponds to application administration and is described in section “Administering a WebTransactions application” on page 127.

5.5.3 Editing a cluster's properties

You can edit the following cluster properties:

- Define the distribution method

  If at least one cluster member is licensed via the cluster then load balancing is automatically used as the distribution method.

  In the case of session roaming (see also section "Roaming Sessions" on page 41), load balancing is also always used as the distribution method since each computer in the cluster is searched for the required session.

  For performance reasons, load balancing does not function with HTTPS servers. It is essential for each computer to possess an HTTP server that permits access to `WTCluster.exe` and `WTPublish.exe`. You can operate this HTTP server very easily in parallel with an HTTPS server and, if necessary, restrict it to the use of `WTCluster.exe` and `WTPublish.exe` by the cluster master.

  Browser access to the cluster is not affected by this and is also possible without restriction via HTTPS.

- Test or delete cluster members
- Add new cluster members to the cluster

- For information on editing cluster properties, see section "Basic operation of the administration program" on page 139.
- For details on editing the properties of a cluster, see the detailed WebTransactions online help system.

5.5.4 Starting a cluster session

Starting a WebTransactions application on a WebTransactions cluster should be transparent for the end user. The user receives a fixed URL which can then be used to start the session in the same way as on a standalone server.

A session on a WebTransactions cluster is started using the following link:

`<a href="http://`*webta-server*`/`*cgi-prefix*`/WTCluster.exe/`*cluster-id*`?[WT_SYSTEM_FORMAT =`*startTemplate*`&`*param2*`....]`

webta-server
Cluster controller on which WebTransactions is running

cgi-prefix
Path at which the CGI programs are stored on the cluster controller

`WTCluster.exe`
CGI program used to control the cluster. This is stored when installing WebTransactions in the script directory of the Web server.

cluster-id
Name of the cluster as specified in the definition.

`WT_SYSTEM_FORMAT` =*startTemplate*
Start template with which the WebTransactions application is started on the cluster member. If no start template is specified in the URL, then WebTransactions uses the template specified in the cluster definition.

`[&`*param2*`....]`
Further optional parameters which can be sent to the WebTransactions application.

Example

In the example below, the cluster `osdtest` is called via its URL. No start template is specified. The start template stored in the cluster definition is used.

```
http://id1windhund/cgi-bin/WTCluster.exe/osdtest
```

Alternatively, you can specify the start template explicitly, in this case `wtstart`:

```
http://id1windhund/cgi-bin/WTCluster.exe/osdtest?WT_SYSTEM_FORMAT=wtstart
```

To start a cluster session from a form you must use the GET method:

```
<FORM METHOD="GET"
      ACTION="http://webta-server/cgi-prefix/WTCluster.exe/cluster-id">
      [<INPUT TYPE=HIDDEN NAME=WT_SYSTEM_FORMAT VALUE="startTemplate">]
  further optional parameters
</FORM>
```

The CGI program `WTCluster.exe` on the cluster controller is responsible for load distribution on the various servers. The name of the cluster is entered as a parameter.

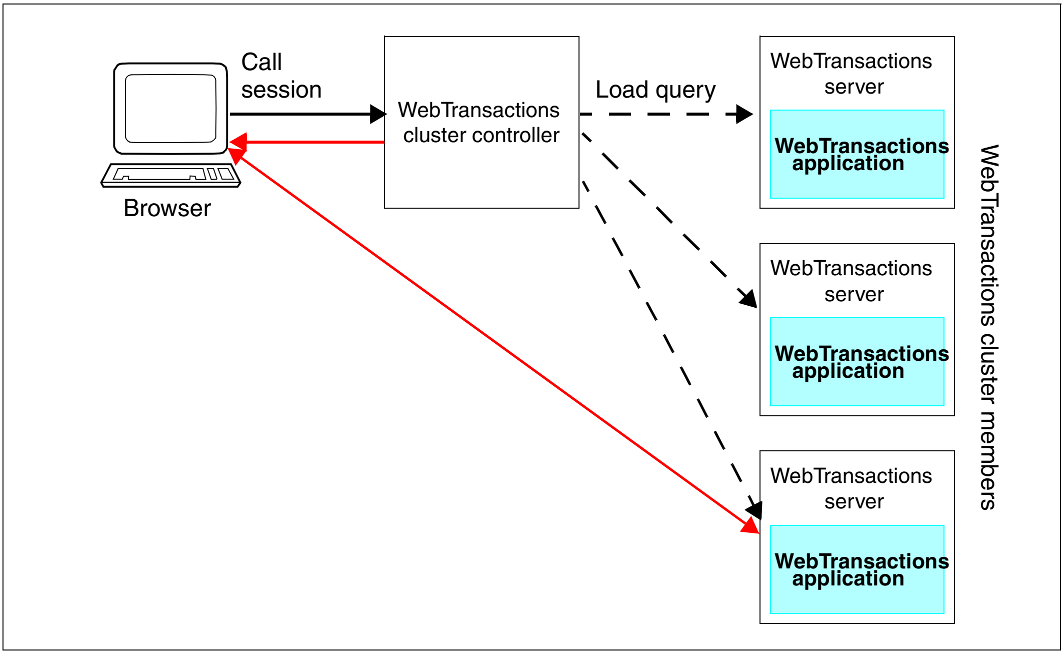

Call session
WebTransactions cluster controller
Load query
Browser
WebTransactions server
WebTransactions application
WebTransactions server
WebTransactions application
WebTransactions server
WebTransactions application
WebTransactions cluster members

Figure 12: Access to a cluster session

`WTCluster.exe` checks the load on the cluster member according to the specified methods and then returns the URL for the start of the “real” session to the browser. Further communication between WebTransactions and the browser takes place directly, or in other words, communication no longer takes places via the cluster controller.

5.6 WebTransactions on a blade server

A blade server is a complex system of independent computer systems (known individually as server blades). A distinguishing feature of blade servers is that they use a small number of optimized hardware components. They reduce the resources required and, as a result, the costs of IT infrastructure. For example, they optimize computer performance for the network and the number of CPUs per volume.

WebTransactions applications treat blade servers as independent computers. They are not like multi-processor computers where the operating system manages the processors. Blade servers, on the other hand, require a cluster software sitting on top of the operating system to handle the logical applications distributed over various blades.

5.6.1 Blade server features

Blade servers have the following features:

- The process modules (the CPU or server blades) are separate from the I/O modules (switch or LAN blades).
- The CPU blades can be individually scaled to match processor performance (e.g. they can be scaled to match the number of users on a terminal server farm or according to the number of client requests on a Web server farm).
- The network connections can be individually scaled to match data throughput.
- CPU blades use the energy saving processors and hard disks developed for notebook computers.
- The compact size of CPU blades and their components ensure high levels of seal.
- Components such as the management blade and the power supply unit are separate from the CPU module.
- CPU blades can be administered, managed and configured remotely over a network. It is also possible to use a local console for administration, management and configuration.

5.6.2 Providing WebTransactions on a blade

This scenario describes how to install and administer WebTransactions on the blade of a blade server and how to enter a license.

This scenario applies to a situation where WebTransactions has to be permanently available on the blade of a blade server with a wide variety of users. You can install WebTransactions applications in this scenario just as you would on any server.

5.6.2.1 Installation on Linux

Installing and configuring the operating system

- Install the Linux operating system.
- During installation, use fixed TCP/IP addresses.
- Start the operating system.
- Once the core operating system is up and running, log on the blade.
- Configure the operating system where necessary (e.g. set up the user IDs, assign user rights, etc.).

Installing and configuring the Web server

- If you have not already done so, install the Apache software. An Apache HTTP server is required software for WebTransactions.
- Configure the Apache system where necessary. You might want to edit the default settings in the `http.conf` file.

  In order to be able to operate WebTransactions, you must disable the `Keepalive` function in Internet Explorer because Internet Explorer does not fully support this function together with Apache. To do this, edit the following line in the `httpd.conf` file:

  ```
  BrowserMatch "MSIE" nokeepalive
  ```
- Start the Web server.

Installing Java (optional)

If you want to call Java objects via WebTransactions, proceed as follows:

- If you have not already done so, install a Java run environment.

Installing WebTransactions

- ► Transfer the compressed archive with the WebTransactions software to the Linux computer.
- ► Log on to the computer.
- ► Install WebTransactions following the description given in the host adapter manual.

5.6.2.2 Installing in Windows

Installing and configuring operating system and Web server

- ► Install the Windows operating system.
- ► If you have not already done so, install the Microsoft Internet Information Server or Apache (see page 150).
- ► During installation, use fixed TCP/IP addresses.
- ► Configure the operating system where necessary (e.g. set up the user IDs, assign user rights, etc.).

Installing Java (optional)

If you want to call Java objects via WebTransactions, proceed as follows:

- ► if you have not already done so, install a Java run environment.

Installing WebTransactions

- ► Create a configuration file for the unattended installation of WebTransactions. The description of the parameters for controlling installation are given in the host adapter manual.
- ► Install WebTransactions as described in the host adapter manual.

5.6.2.3 Configuring WebTransactions

You can use the browser of your choice to administer and configure WebTransactions.

- ► Start the administration program using the following URL: `http://`*webta-server/cgi-präfix*`/WTPublish.exe/server-admin`. *webta-server* is the name of the computer on which WebTransactions is currently running, possibly with port indication. *cgi prefix* is the path on this computer for the CGI program.
- ► Following your usual practices, enter a license key, set up users and pools and assign users pools (see section “Managing the WebTransactions server” on page 139).

5.6.3 Providing WebTransactions on several blades of a blade server

This scenario describes how you can make WebTransactions quickly available on several blades with the help of an image. This scenario describes the automatic entry of IP addresses, server names, and the entry of licenses on cloned blades.

This scenario is designed for situations where WebTransactions is to be used for a high load host application or for many separate host applications with a high total load running on a blade server.

Installing the reference blade

- ► Install WebTransactions as the reference application on a blade.
  Follow the instructions given in the previous section “Providing WebTransactions on a blade” on page 150.

Creating an image

- ► Create an image of the installation.
  In the sections which follow, this image is referred to as the WebTransactions core image.

Making clones

- ► Distribute the WebTransactions core image over several blades.
- ► Use the same, fixed TCP/IP addresses on the target blades (production blades).

Configuring WebTransactions

The production blades have different names and IP addresses from reference blades. During the cloning procedure, the license key of the reference blade is copied onto the production blade. However, this license key is not suitable for identification of the production server. You must therefore use WebTransactions Administration to obtain and then enter a suitable license key (see section “Managing the WebTransactions server” on page 139).

5.6.4 Providing a WebTransactions cluster on several blades of a blade server

This scenario is similar to the second scenario above with the difference that in this case you will be distributing a ready-to-use WebTransactions application and a cluster definition over several blades; you use the image procedure as before.

This scenario is designed for situations where a single application runs with many, many clients and as a consequence the load has to be distributed over many blades. Installation in this case is made considerably easier by the image procedure because this automatically adjusts the number of installed blades to the size of the load.

Configuring the applications, setting up the cluster and distributing the application

WebTransactions is designed to be easy to develop, configure and administer via http connections. There are no restrictions or special instructions even where blade servers are used. The procedure is the same as on other platforms and heterogeneous clusters. Configure your application, set up the clusters and then distribute the master application to the cluster members (see section “Cluster concept” on page 142).

Saving a dynamic load to clusters

In many Web applications there are regular variations in loads. In order to be able to guarantee good performance during peak periods, you must set up a specific cluster with a sufficient number of blades to take the load. However, it often makes sense to assign other tasks to these blades for use in periods when the loads are lower. To do this enter the necessary images on the blades.

Proceed as follows:

- ► Install WebTransactions on the maximum necessary number of blades as described in the section “Providing WebTransactions on several blades of a blade server” on page 152.
- ► On one or more of the blades which will remain permanently installed, create a definition for a WebTransactions cluster; this cluster will be responsible for distributing requests with the `Take first with load lower x%` method (see section “Editing a cluster’s properties” on page 146).

  The `Take first with load lower x%` method makes sense here because it does not take into account cluster members which are not currently active. You can use this procedure to shutdown blades. The cluster continues to work without problems as long as at least one CPU blade is active.
- ► Enter all the blades (giving a computer name or IP address) involved in the operation of a cluster as members of that cluster.

- Next, use WebLab to distribute the WebTransactions application to all the blades.
- Save an image of all the blades that will be undertaking other tasks at various time (see "Creating an image" on page 152).
- Later recreate these blades using the image saved. This means you will not have to enter the license key again because the matching key has already been correctly entered in the saved image.

6 The WebLab development environment

WebLab is the WebTransactions development environment. It is designed to help you create WebTransactions applications, edit the templates, and view the results. It is used in conjunction with any Web browser, in which you can check the effects of your changes immediately (WYSIWYG).

WebLab also offers an import/export interface to an HTML editor of your choice. All HTML tags can be inserted and edited conveniently under the WebLab interface. WebLab offers wizards for inserting and editing the most common HTML tags.

WebLab, the HTML editor, and the browser run on a PC under Windows.

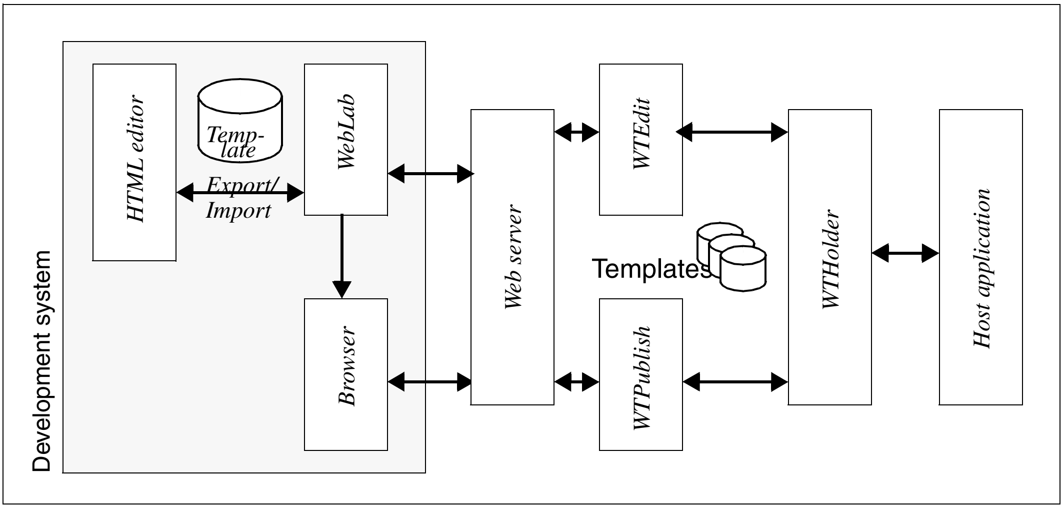


Figure 13: The architecture of WebLab

WebLab and the Web browser are connected and coordinated via a WebTransactions session running in the background. This allows you to adapt WebTransactions applications “on the fly”, and to check or explicitly reset current values and settings.

WebLab

WebLab is the central coordination point of the development environment. It starts and manages a WebTransactions session, which runs in the Web browser. WebLab can load and save the files (templates and administration files) of this WebTransactions session, and allows the user to access, view, and if necessary modify the objects of the current session.

Web browser

The browser defined using the **Options/Preferences** command is used for navigating around the WebTransactions application and displaying templates. The concept of an integrated WYSIWYG component in WebLab was deliberately rejected because the appearance of an HTML page can change depending on the browser used. You can use the same browser for development purposes as provided for normal operation. Browser-specific extensions are also supported. To display the current template in the browser, all you need to do is click on a WebLab menu item. Changes in the layout of a page can thus be displayed immediately without having to repeat the dialog steps originally performed to access this page.

HTML editor

As an option, WebLab can be extended to include an HTML editor of your choice.

However, the selected HTML editor must meet the following requirements:

- HTML comments must remain unchanged.
- Unknown tags should not be corrected automatically, or it must be possible to disable this functionality.

WebLab provides functions for editing templates used for working in the HTML editor. The HTML editor and WebLab are connected by means of an export/import mechanism.

6.1 Functionality of WebLab

To help you integrate your host application and modernize your interfaces, the WebLab functionality includes the following features:

- Simple, fast integration of existing host applications by:
  - creating projects
  - creating a base directory
  - automatically generating the required templates
  - providing support when creating an application-specific start template
- A user-friendly means of editing templates by:
  - providing support when entering and modifying WTML tags via dialog boxes for parameter specifications.
  - the various language resources that you can use in a template are marked in different colors for rapid identification.
  - providing support when entering and modifying HTML tags via dialog boxes for parameter specifications.
  - providing support when entering and modifying WTBeans through dialog boxes for specifying parameters. WTBeans already contained in the template can also be edited: this is carried out using the same dialog box as when entering the WTBean. However, in this case, the fields are preset to the current values. The various components of the WTBeans are displayed against different color backgrounds to permit rapid identification.
  - providing support during the import of comments and the possibility to create HTML documentation for the templates from the comments.
  - granting direct access to the objects of the WebTransactions session. In WebLab, these objects are displayed in a separate window. Object and attribute names (e.g. for host data objects originating from the host application) can be inserted in templates using the mouse.
  - permitting the monitoring of variables in a separate window.
  - allowing you to select host data objects in a graphical representation of the original screen.
  - enabling you to open referenced includes and class templates within your template at the click of a mouse.

  - offering wizards for the provision of typical graphical dialog control techniques (e.g. drop-down lists or radio buttons). Another wizard offers support when creating a template from an HTML page. WebLab inserts the WTML language elements required for data exchange and communication with the host in the HTML page.

- Support during the testing of a WebTransactions application through
  - template design tests (see section “Testing the design of a template” on page 191)
  - single step tracking (see section “Testing the execution sequence in the template” on page 193)
  - special start templates
- Enhanced security through close cooperation with the administration program
- Simple transfer of entire WebTransactions applications

6.2 First Steps

WebLab runs on a Windows system. However, WebTransactions and thus the integration server and the host application can run on other platforms. It is possible to integrate the host application remotely without having to log in to the integration server. The required data is supplied via the network on the basis of the HTTP or HTTPS protocol.

The following sections describe how to integrate your host application quickly and easily in the WWW.

The general concept of postprocessing is described in section “Editing templates” on page 170. A sample session including a postprocessing phase can be found in the manuals for the individual WebTransactions supply units.

6.2.1 Creating projects

The project is the main entry point into WebLab. The project file contains the most important data that WebLab needs when working with a WebTransactions application, for example the WebTransactions server data. You use a project for the permanent storage of the individual session settings for a WebTransactions application.

All the commands used to create and administer projects can be found in the **Project** menu.

You can configure WebLab in such a way that the last project used is loaded immediately when WebLab is started.

For a detailed description of all the project-related functions available in WebLab, refer to the detailed WebLab online help system. To do this, choose the command **?/Contents**.

- To create a new project, choose the command **Project/New...** and then click on **Yes** when you are asked whether you really want to create a base directory.

  The **Connection to Server** dialog box opens.

- You should then proceed as described in the following sections:
  - To create a base directory, see page 160
  - To create an automask template, see page 161
  - To generate an individual start template, see page 161

For detailed instructions, refer to the manuals corresponding to the individual WebTransactions supply units and the detailed WebLab online help system.

6.2.1.1 Creating a base directory

The most important steps involved in integrating a host application are creating a base directory and opening a session.

The base directory is created on the system on which WebTransactions runs. For this purpose, WebLab establishes a connection with the WebTransactions host and starts the CGI program `WTEdit` there. `WTEdit` reads the necessary information from the WebTransactions installation directory and uses it to create a base directory.

Please note that users and at least one pool must be set up for WebTransactions applications beforehand using the WebTransactions administration program. For further information, please refer to chapter “WebTransactions server” on page 131.

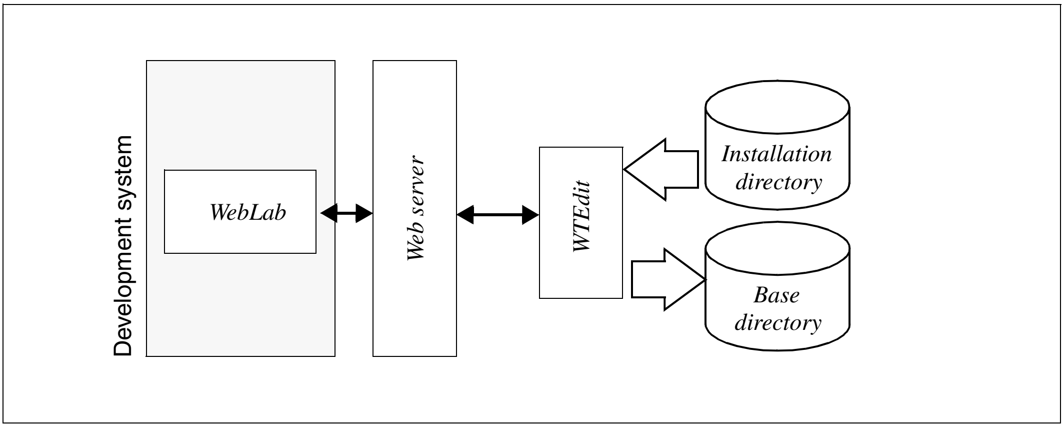


Figure 14: Procedure for creating a base directory

► Establish the connection to the server (**Connection to Server** dialog box).

To do this, specify the parameters required in order to open a connection (for **URL of WTPublish** and **URL of WTEdit**).

► Now enter the properties of your base directory (**Create Basedir** dialog box).

A new base directory is now created under the entered name in the selected pool.

6.2.1.2 Creating an automask template (OSD, MVS)

- As soon as you confirm the **Create Basedir** dialog box with **OK**, the **Automask generation** dialog box is displayed.

  The automask template controls the automatic conversion of formats into a browser-compatible form for host applications using the 9750 (OSD) and 3270 (MVS) protocols, i.e. the formats are emulated directly in the browser without any capture or postprocessing.

- The automatic conversion process can be defined using the parameters in the **Automask generation** dialog box:

  If you wish, you can then create a start template for one of the communication objects selected during generation.

6.2.1.3 Creating an individual start template

You can generate an individual start template for your host application immediately. WebLab then opens a dialog box in which you can make specifications concerning the connection to your host application.

The **connection parameter** tab contains all the mandatory settings, e.g. the name of the host application or the name of the computer on which the host application runs.

6.2.2 Saving a project

You can save a project on your hard disk at any time (**Project/Save** or **Save as...**).

If the project has not been saved when you close the project or WebLab then you are asked if you want to save the project.

6.2.3 Starting a session

- Select **File**/**Start Session** to open the **Start Session** dialog box.
- In this dialog box, you can enter the parameters required to open a session:
  - Here, you can edit the server connection parameters via **URL of WTPublish**.
  - You can also specify other parameters and settings that are of relevance for the session, e.g. base directory, start template etc.

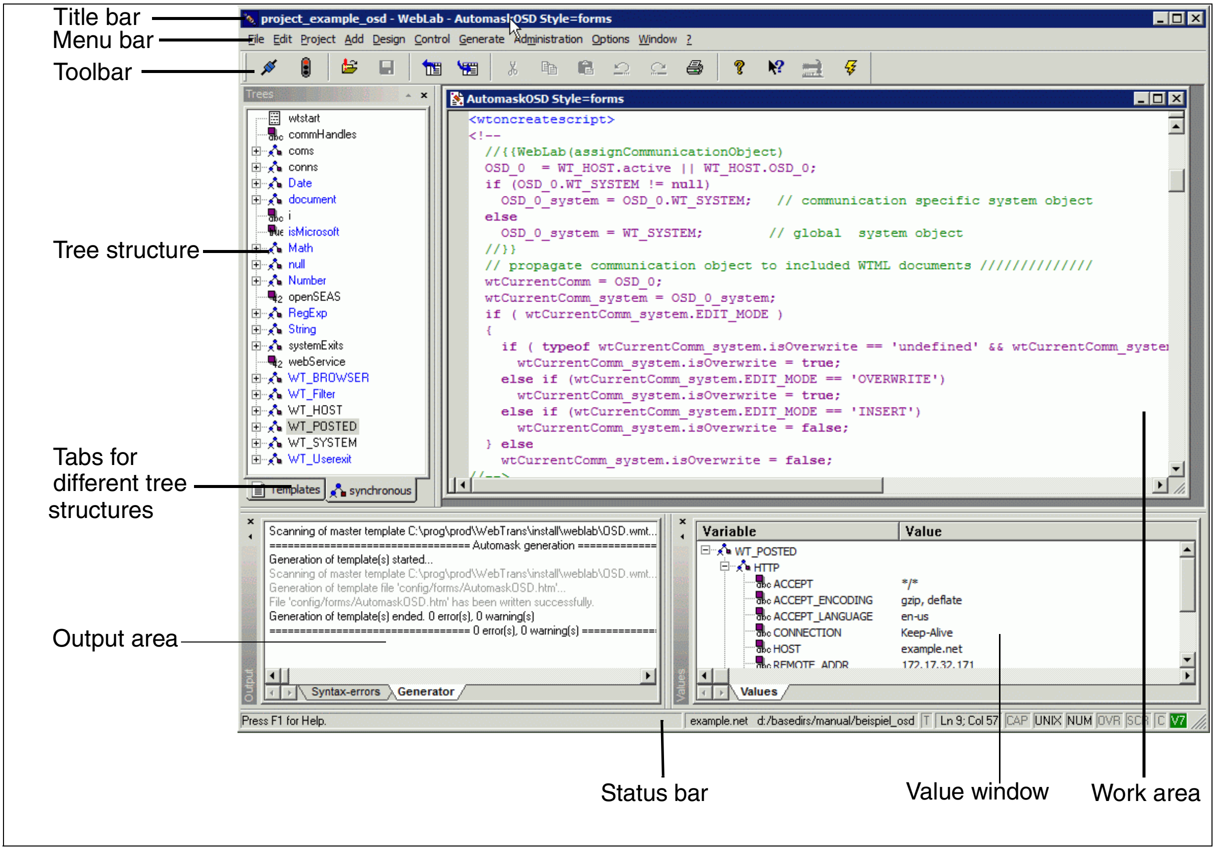
⇨ For detailed instructions, refer to the manuals corresponding to the individual WebTransactions supply units and the detailed WebLab online help system.

6.3 The WebLab GUI

This section contains a brief introduction to the WebLab GUI. For further information on WebLab, please refer to the WebLab online help system.

6.3.1 Main window

As soon as you start WebLab by selecting **Start/Program Files/WebTransactions 7.5/WebLab**, the following main window appears:



```
<wtoncreatescript>
<!--
  //{{WebLab(assignCommunicationObject)
  OSD_0  = WT_HOST.active || WT_HOST.OSD_0;
  if (OSD_0.WT_SYSTEM != null)
    OSD_0_system = OSD_0.WT_SYSTEM;   // communication specific system object
  else
    OSD_0_system = WT_SYSTEM;        // global  system object
  //}}
  // propagate communication object to included WTML documents //////////////
  wtCurrentComm = OSD_0;
  wtCurrentComm_system = OSD_0_system;
  if ( wtCurrentComm_system.EDIT_MODE )
  {
    if ( typeof wtCurrentComm_system.isOverwrite == 'undefined' && wtCurrentComm_syste
      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;
//-->
```

```
Scanning of master template C:\prog\prod\WebTrans\install\weblab\OSD.wmt...
================================ Automask generation ============
Generation of template(s) started...
Scanning of master template C:\prog\prod\WebTrans\install\weblab\OSD.wmt...
Generation of template file 'config/forms/AutomaskOSD.htm'...
File 'config/forms/AutomaskOSD.htm' has been written successfully.
Generation of template(s) ended. 0 error(s), 0 warning(s)
================================ 0 error(s), 0 warning(s) ============
```

Variable	Value
WT_POSTED	
HTTP	
ACCEPT	*/*
ACCEPT_ENCODING	gzip, deflate
ACCEPT_LANGUAGE	en-us
CONNECTION	Keep-Alive
HOST	example.net
REMOTE_ADDR	172.17.32.171

```
Press F1 for Help.    example.net  d:/basedirs/manual/beispiel_osd  T  Ln 9; Col 57  CAP UNIX NUM OVR SCR C V7
```

Figure 15: Main WebLab window

The main window consists of a title bar at the top containing the name of the current template, followed by a menu bar and a toolbar.

It is divided into two areas:

- The left-hand side contains various **tree structures** in which the objects of the current template and the files in the base directory are arranged in a hierarchy similar to that of the Windows Explorer.

  As soon as a connection is established, the template tree is displayed with the files present in the base directory of the WebTransactions application. Otherwise the tree structure is empty.

  If a session is started, the object tree of the template that is currently being executed is displayed in the tree structure.

- The right-hand side contains a **work area** in which the templates are displayed for editing. This area can also be used to view the output from the template browser. When WebLab is started, the work area is initially empty.

- During your work with WebLab, status or error messages are output in various tabs in the **output area** below the work area:

  - for generator output (e.g. when generating the base directory or format-specific templates)
  - for error messages
  - for WebTransactions application distribution
  - for the display of variables and values during single step tracking
  - for outputting tools at the server

  You can switch between these windows using the index tabs.

- The lower area also contains the **value window** in which variables, objects and the associated values are displayed.

At the bottom of the main window is a status bar. This displays a short, context-sensitive help text, and indicates the connected host and the base directory used. When you edit a template, the line and column number in which the cursor is currently positioned in the template are also displayed.

6.3.2 Tree structure

At the bottom of the tree structure, you will see a row of index tabs which can be used to switch between the various tree structures. The number of tabs that are displayed depends on the current session.
The following tabs are possible:

Templates
Tree structure of the base directory

synchronous
Object tree of a synchronous dialog

asynchronous
Object tree of an asynchronous dialog

remote
Object tree for accesses via WT_REMOTE

6.3.2.1 Template tree

As soon as a connection is active, all the files present in the base directory are displayed in a tree structure. This tree structure is handled in the same way as the Windows Explorer: you can expand and collapse the different hierarchical levels. A filter allows you to exclude certain files from the display. To select individual files, simply click on them with your mouse.

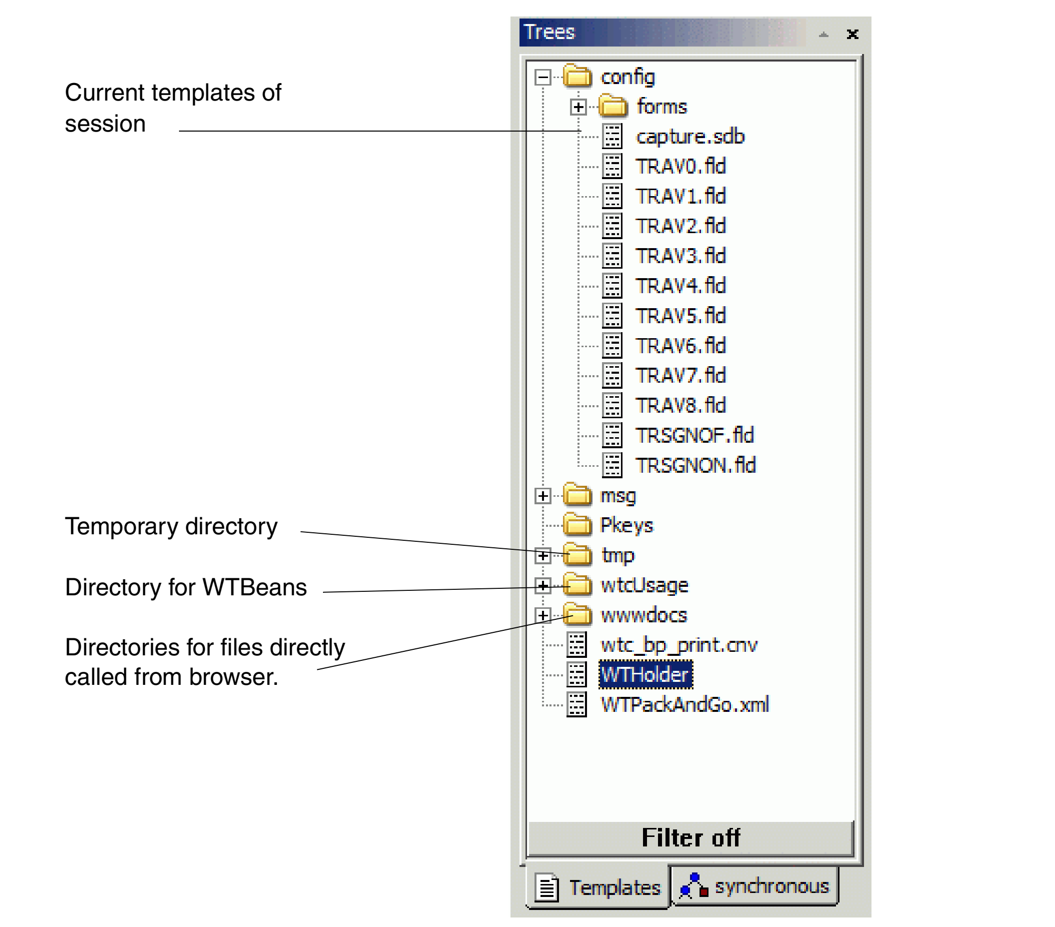


Figure 16: Template tree in WebLab

Context menu

If you select a file in the tree structure and click the right mouse button, a context menu is opened containing commands specific to the current file.

You can, for example, activate the template browser via the context menu (**Show Context**). This presents the various relations between the templates in graphical form.

⇨ For a detailed description of the individual menu items, see the detailed WebLab online help system.

File selection

There are two ways of filtering the display in the template tree:

- by filename suffix
  Files are displayed either if they have one of the specified suffixes or if they do not have any of the specified suffixes
- by creation date
  Files are displayed depending on whether their creation date lies before or after the specified date. You can also specify a period (from – to).

You can combine the two filter modes using AND or OR operators.

In this way, it is possible, for example, to display only files with the suffix `.bak` that are more than one month old.

You can use the **Filter off/on** button below the template tree to modify the filter settings and display the active filters.

⇨ For information on setting up new filters or editing existing ones, see the detailed WebLab online help system.

6.3.2.2 Object trees

As soon as a session is started, all the objects and variables of the current session are displayed in a tree structure.

Variables that are available globally for a WebTransaction session, are marked in blue. These are the attributes that have been imported from WebTransactions modules. In the **Object Tree** tab (command **Options**/**Preferences**), you can modify the way the object tree is sorted and hide global session attributes from the object tree.

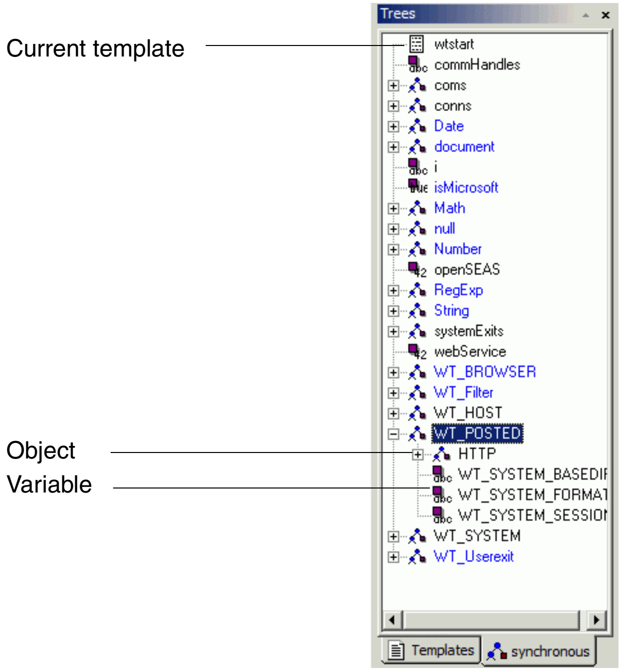


Figure 17: Object tree in WebLab

The values of objects and variables can be viewed and modified using the context menu **Properties** command. The objects and variables themselves can be inserted in the current template using the Drag&Drop mechanism. You use the following key combinations to define what is inserted in the template:

Drag the mouse
Inserts the full variable name in the template.

Drag while holding down the Shift key
Inserts the attribute name in the current template.

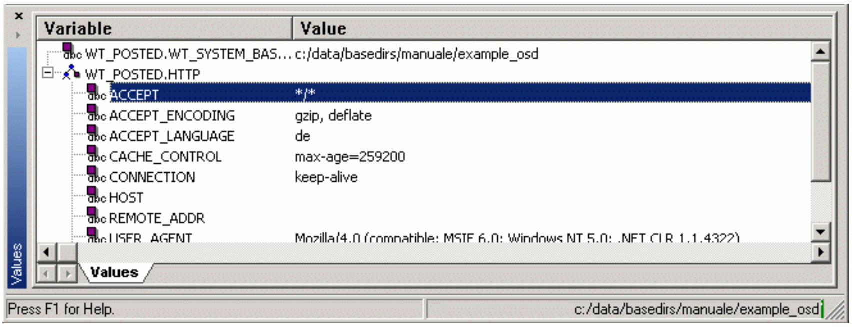
Drag while holding down the Alt key
Inserts the variables with the evaluation operator in the current template.

Drag while holding down the Ctrl key
Inserts the variable value in the current template.

6.3.3 The value window

The value window, which takes the form of a separate, dockable window, displays variables and their associated values. You can drag variables and objects from the object tree and the graphical host object selection into the value window using the mouse.

The objects are displayed in a two-column list as depicted below



Variable	Value
WT_POSTED.WT_SYSTEM_BAS…	c:/data/basedirs/manuale/example_osd
WT_POSTED.HTTP	
ACCEPT	*/*
ACCEPT_ENCODING	gzip, deflate
ACCEPT_LANGUAGE	de
CACHE_CONTROL	max-age=259200
CONNECTION	keep-alive
HOST	
REMOTE_ADDR	
USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)

Press F1 for Help. c:/data/basedirs/manuale/example_osd

In the case of objects that have an associated value, the name and path of the variable are displayed in the left-hand column and the value in the right-hand column. If you double-click on a value then this is displayed in a separate window in which line breaks are inserted at a suitable point in long values

In the case of variables that have a structure, only the name of the variable is displayed in the left-hand column together with a + character which you can click on to open the structure.

➪ For information on working with the value window, see the detailed WebLab online help system.

6.4 Generating templates

WebLab allows you to generate the following templates automatically:

- Automask templates (OSD, MVS)
- Format-specific templates by means of the capture procedure (OSD, MVS)
- Format-specific templates from IFG descriptions (OSD, openUTM)
- Templates for simple access to web services

➪ For a description of the detailed procedure, see the manuals accompanying the various WebTransactions supply units.

6.5 Editing templates

Templates can be edited directly in WebLab without having to be transferred explicitly to the development PC. This automatically takes care of the development environment for you.

6.5.1 General procedure

To edit templates on the server, proceed as follows:

- ► Select **File**/**Start Session** to open the **Start Session** dialog box.
- ► Confirm the dialog box with **OK**. WebLab then starts the Web browser for running the WebTransactions session. Once the session is started, the object tree is displayed. This is reloaded from the server each time the current template is loaded or the **Control/Update in Browser** command is selected.
- ► In the browser, navigate to the screen format you wish to edit.
- ► Load the associated template into WebLab by selecting **File/Open Current Template**.
- ► Edit the template as desired.
- ► Select **Control/Update in Browser** to view the effect of your changes. WebLab then updates the template in the browser.

i For a detailed description of all WebLab functions see the detailed WebLab online help system.

6.5.2 Designing templates

The user interface offered by the format-specific templates reflects the appearance and functionality of the terminal display. If you want to change the graphic presentation of the template and, for example, convert the menu into a drop-down list, then you must edit the templates. You can do this using either the normal resources for web page design or the WTML template language which is described in detail in the WebTransactions manual “Template Language”.

6.5.2.1 Editing templates

There are no restrictions to the individual adaptations you can make to your templates: you can insert animated or clickable images, Java scripts and applets, ActiveX controls, video and sound sequences etc. You can use all the language resources that are supported by the intended Web browser.

WebLab provides you with support for inserting and editing the most important HTML tags (**Add/HTML or Edit/Edit Tag** commands). You can influence the notation used for the inserted Tags (**Options**/**Preferences**/tab **Format** command). If you choose the option **Create HTML tags in XHTML form**, then WebLab creates all HTML tags XHTML conform.

However, you can also design the HTML user interface with an HTML editor independently of the generated templates and then use the WebLab **Design/Merge** command to convert this into template form.

With WTBeans, WebTransactions provides you with a set of reusable components which you can use to control the display of your data in the browser and control communication with the host application. For more information, see the next section and section “WTBeans” on page 53.

You can use the commands in the **Insert**/**WTScript** menu to insert prototypes for all WTScript functions and classes in a template.

6.5.2.2 Insert snippets

Snippets are text components, which can be taken over unchanged in templates. Snippets are stored together with user specific data in your computer.

To display previous snippets, choose the command **Options**/**Show Snippets**. By double clicking on the corresponding snippet it is inserted in your template on the cursor position.

Also the context menu in the snippets window offers functions to edit snippets, rename, delete, as well as to create new folders and new snippets.

6.5.2.3 Inserting WTBeans

The commands differ depending on whether you want to insert a standalone or inline WTBean.

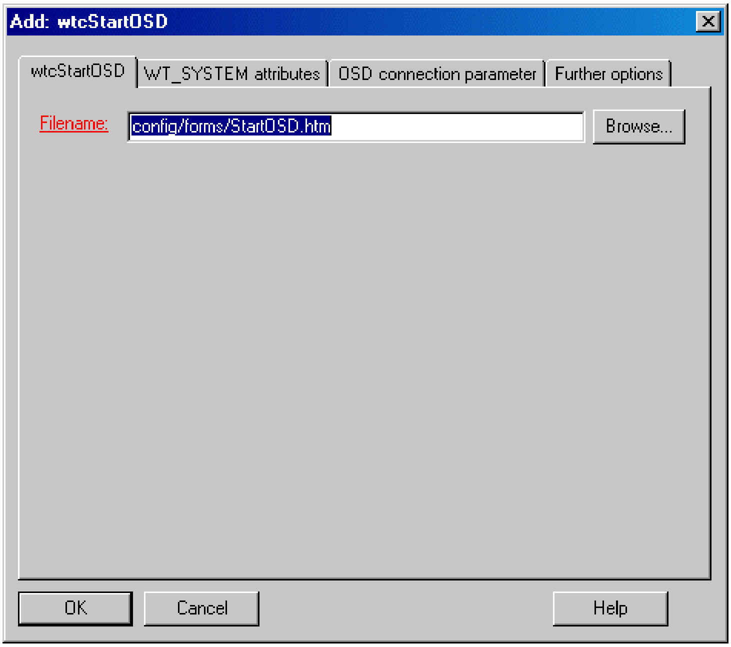
i Before you can insert or edit WTBeans, a connection to the WebTransactions server must be established.

When you download WebTransactions, you can also retrieve additional WTBeans as goodies. These additional WTBeans must be installed separately. For information, see the associated documentation. After installation, the additional standalone
WTBeans are also displayed in the submenu.

Standalone WTBeans

- ► To insert a standalone WTBean you must first connect to a WebTransactions application.
- ► To do this, select the **File/New** command in WebLab. A list of WTBeans for selection is then displayed in a submenu. The standalone WTBeans for the creation of a start template are supplied with the product.

WebLab provides a uniform GUI in which you can edit the parameters of all WTBeans. This GUI is depicted here using the example of the supplied WTBean `wtcstartOSD`.



Filename: config/forms/StartOSD.htm

You can edit the available parameters in one or more tabs of the displayed dialog box. The names of the parameters for which the entry of a value is mandatory are displayed in red. Default values are entered for all the other parameters if you do not specify anything.

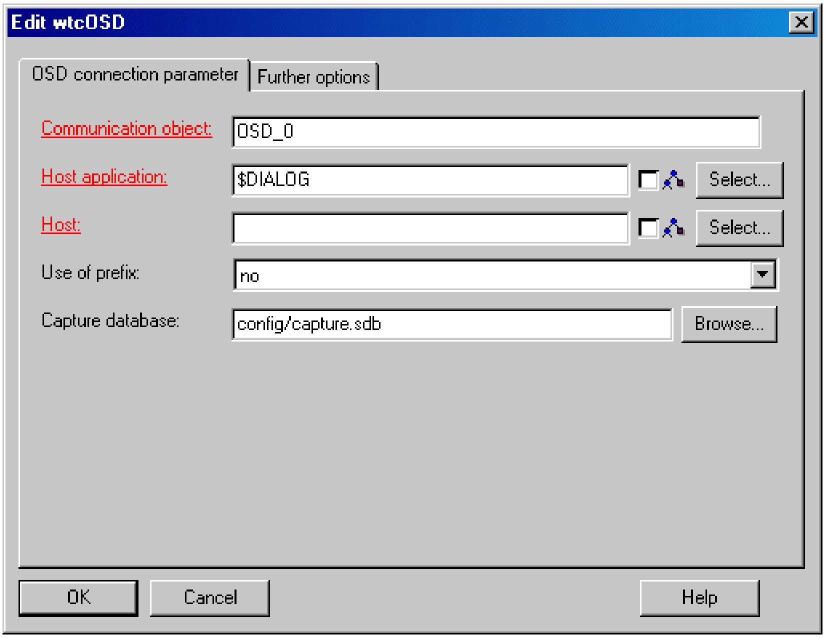
- ► Confirm your specifications with **OK**. The corresponding template is generated on the basis of your specifications and the description file and is displayed in the WebLab work area. For more information, refer to section “Displaying WTBeans in the template” on page 174.

The generated start template is saved under the path that you enter in the first WTBean tab. The description file for the used WTBean is stored in the `wtcUsage` subdirectory of the base directory.

Inline WTBeans

- ► To insert an inline WTBean, first open the template in which you want to insert the WTBean in WebLab.
- ► Move to the corresponding position in this template and choose the **Insert/WTBean** command. A list of WTBeans for selection is then displayed in a submenu. The inline WTBeans for the creation of a communication object for the various host adapters are supplied with the product.

  WebLab provides a uniform GUI in which you can edit the parameters of all WTBeans. This GUI is depicted here using the example of the supplied WTBean `wtcOSD`.



Communication object: OSD_0

Host application: $DIALOG □

Host: □

Use of prefix: no

Capture database: config/capture.sdb

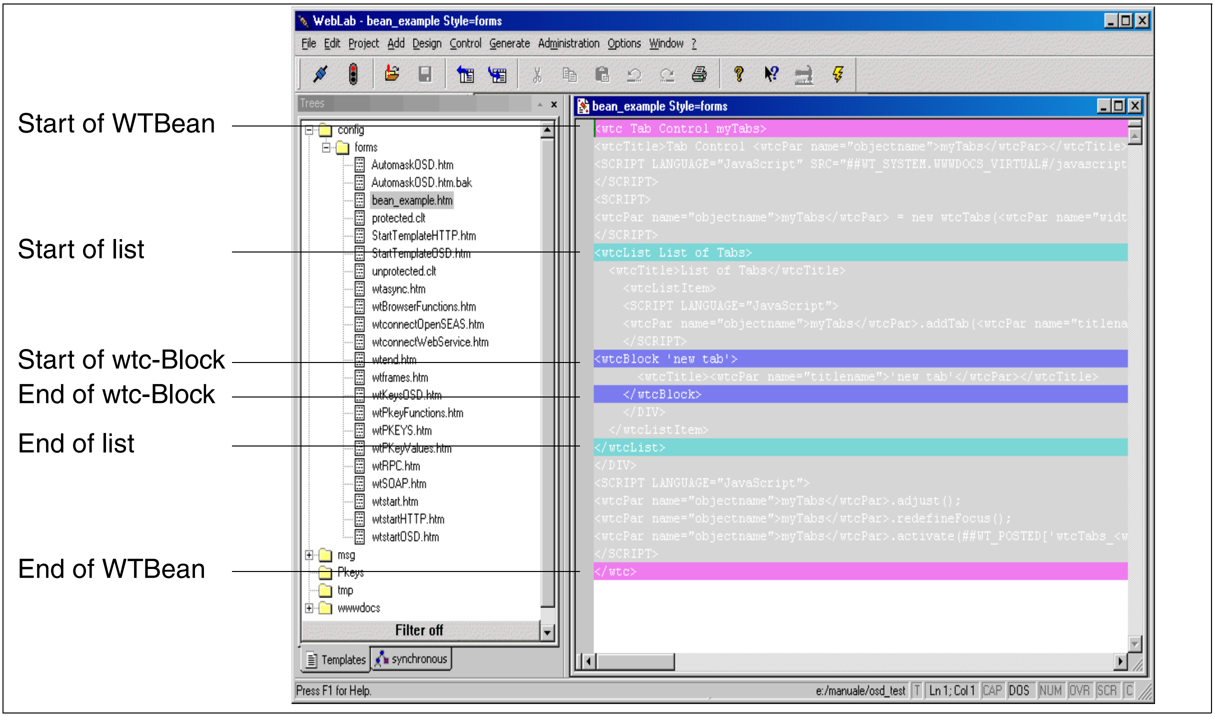
You can edit the available parameters in one or more tabs of the displayed dialog box. The names of the parameters for which the entry of a value is mandatory are displayed in red. Default values are entered for all the other parameters if you do not specify anything.

- ► Confirm your specifications with **OK**. The corresponding WTML code is generated on the basis of your specifications and the description file and is inserted in the template. The description file for the used WTBean is stored in the `wtcUsage` subdirectory of the base directory.

Displaying WTBeans in the template

You can insert and edit WTBeans in WebLab. For precise information on how to do this, refer to section “Inserting WTBeans” on page 171. A WTBean consists of the code itself and blocks of freely definable HTML text in which you can insert texts, scripts or WTBeans of your choice.

You cannot edit the code of a WTBean, irrespective of whether it is a standalone or inline WTBean. For this reason, this area is displayed in gray in the template. The start and end lines of a WTBean are displayed on a pink background. You can subsequently edit the properties of a WTBean. For more information, refer to section “Editing WTBeans” on page 175.



```
<wtc Tab Control myTabs>
<wtcTitle>Tab Control <wtcPar name="objectname">myTabs</wtcPar></wtcTitle>
<SCRIPT LANGUAGE="JavaScript" SRC="##WT_SYSTEM.WWWDOCS_VIRTUAL#/javascript
</SCRIPT>
<SCRIPT>
<wtcPar name="objectname">myTabs</wtcPar> = new wtcTabs(<wtcPar name="widt
</SCRIPT>
<wtcList List of Tabs>
 <wtcTitle>List of Tabs</wtcTitle>
  <wtcListItem>
  <SCRIPT LANGUAGE="JavaScript">
  <wtcPar name="objectname">myTabs</wtcPar>.addTab(<wtcPar name="titlena
  </SCRIPT>
<wtcBlock 'new tab'>
   <wtcTitle><wtcPar name="titlename">'new tab'</wtcPar></wtcTitle>
  </wtcBlock>
  </DIV>
 </wtcListItem>
</wtcList>
</DIV>
<SCRIPT LANGUAGE="JavaScript">
<wtcPar name="objectname">myTabs</wtcPar>.adjust();
<wtcPar name="objectname">myTabs</wtcPar>.redefineFocus();
<wtcPar name="objectname">myTabs</wtcPar>.activate(##WT_POSTED['wtcTabs_<w
</SCRIPT>
</wtc>
```

The areas of a WTBean in which you can enter text and other WTBeans are identified by a so-called `wtc` block the start and end of which are displayed against a blue background.

The WTBean description file also defines what can be inserted in this block:

- If you are able to insert text in the block then you can enter blank lines after the start of the wtc blocks.
- If you are able to insert further WTBeans then these are displayed in the submenu or in the wtc block’s context menu when you choose the **Insert/WTBean** command.

Editing WTBeans

Even though you cannot edit the code of a WTBean, you can still edit its parameters.

- ► To do this, right-click on the start line of the WTBean to open the context menu.
- ► Choose the **Edit WTBean** command. The dialog box with the tabs for parameter editing is opened. The parameters are preset to their current values.

You should note the following changes in WebLab responses when you edit templates that contain WTBeans

- You can only select the entire WTBean: when you select a line within a WTBean, the whole WTBean is selected. The only exception here is the text in a wtc block.
- The non-editable areas (gray background) are ignored during search and replace operations.

6.5.3 Defining templates for host formats

In this section, you will find in-depth information on the following subjects:

- Defining a global layout that applies to all templates
- Designing host formats
- Graphical selection of host objects

6.5.3.1 Defining the global layout

There are a number of different ways of implementing a global layout that applies to all templates:

- Include templates
- Master templates
- The system object attributes `EPILOG`, `FORMTPL` and `PROLOG`

Include templates

If you want your design measures to affect all the templates, for example if you want to insert company logos or make general information available on all your pages, then you can write the corresponding passages in a separate file and insert these in your templates by means of include tags. This simplifies the task of template maintenance. If something changes then you need implement the modification only once in the central Include template.

To include files in WebLab use the command
Add/Include... or
Add/WTScript/Template Functions/include, if you are in a script area.

Master templates

You can also define the global layout via master templates.

You specify the master template that is to be used for generation in the WebLab **Options for FLD and Template Generation** dialog box during template generation. Some generation options (e.g. the method used for generation) can be defined both in the master template and directly using WebLab.

In this case, the settings in the master template are taken over as the default values in the dialog box. You can edit them here with the result that the modified values override the corresponding definitions in the master template. This means that the values displayed in the dialog box are used whenever a template is generated.

You can find a detailed description of the master templates in the corresponding chapter of the WebTransactions manual “Template Language.

The system object attributes `EPILOG`, `FORMTPL` and `PROLOG`

You can also use the attributes `EPILOG`, `FORMTPL` and `PROLOG` of the connection-specific system object to influence global template design.

The attributes each contain the name of a template that is executed at different times depending on the attribute in question:

`PROLOG`	at the start of the current template
`FORMTPL`	before execution of the DataForm tag at the end of the current template
`EPILOG`	at the end of the current template

The supplied master templates ensure that the templates specified in the attributes are included in the generated templates.

6.5.3.2 Design host formats

You can use the WebLab wizards to perform standard beautification steps automatically. These wizards replace generated input fields (INPUT tags of type “text”) with graphic elements such as drop-down lists, radio buttons, checkboxes and pushbuttons. (You will find an associated example session in the manuals for the various supply units).

You can call the wizards from either the **Design** menu or the context menu for graphical host object selection.

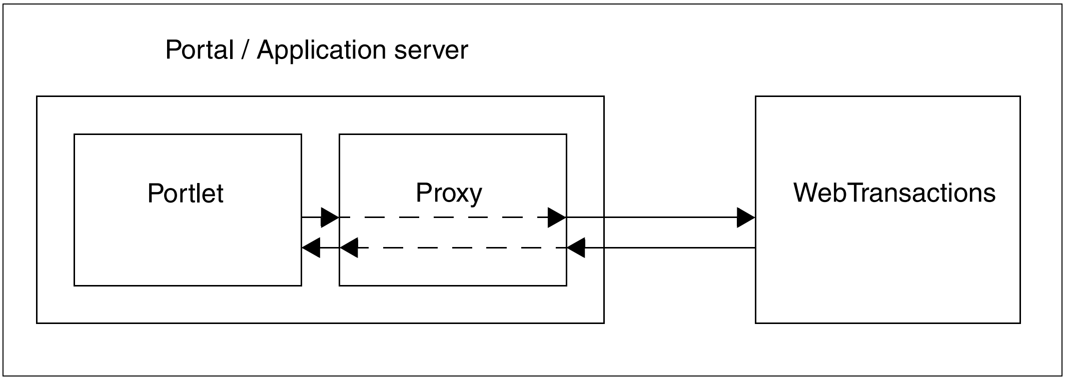
6.5.3.3 Select host objects graphically

Since it is not always possible with host data object names to obtain a conclusion for the use of the object, WebLab offers you for these objects an additional graphic selection possibility (**Design/Select Host Objects Graphically** command): in a display of the original object, you can choose a host data object by clicking the corresponding interface element. With a double click, the corresponding name is inserted in the current cursor position in the edit area.

Through Drag&Drop completely qualified variable names can also be moved into the work area. If you push the `Shift` key during drop and drag, then only the attribute name is used. If you hold down the `Alt` key, then the fully qualified name is inserted within an evaluation operator. Hold down the `Ctrl` key, to insert the current value of the variable.

6.5.4 Designing templates for portal use

Internally, the portlets or iViews that support the integration of host dialog applications in portals use a proxy to communicate with the WebTransactions application:



This proxy performs portal-specific modifications to generated pages before they are displayed in the browser. This ensures that references (e.g. to images) are not resolved outside of the portal. Due to this design and because the portlets run on only a part of the page in the portal, the following constraints apply to template design:

Constraint	Applies to	
	Oracle/ JSR-168 Portlets	**SAP EP iViews**
Do not use the whole browser window for output: – In scripts, do not use any constructs such as `top.location = ...` – Do not set `target="_top"` in links.	X	X

Constraint	Applies to	
	Oracle/ JSR-168 Portlets	SAP EP iViews
The proxy only forwards the http header "User-Agent".	X	
Substitutions in the responses are made only if the content type is text/html or text/javascript.	X	
Any addresses present in the document are adapted in such a way that they address the proxy. Addresses are recognized in the href attribute of a and link tags, in the src attribute of the frame, iframe, img and script tags, as well as in the action attribute of the form tag. Addresses in Javascript are only recognized as such (and modified) if they are present in assignments (after a =) and if the variable to which they are assigned ends with `document.location`, `document.location.href` or `.src`.	X	
There must be precisely one form tag in each template form; this must address the WebTransactions application.		X

6.5.5 Designing dialog sequences

WebTransactions does not just allow you to “face lift” your host applications. You can also redesign your dialog sequences. The inflexible 1:1 correspondence between HTML page and host format is a thing of the past. With WebTransactions you can actively control and modify the dialog strategies implemented by the host application: input and output elements can be filtered out or added while dialog steps can be fused or subdivided.

Recording dialogs in WTScript

WebLab allows you to record dialog steps with your host application in the form of a WTScript. In this way, you can, for example, combine multiple dialog steps with the host application in a template. You can record the entire sequence of dialog steps, insert them in the template and then run them.

To record the dialog steps, you use the commands in the **Control**/**Dialog Recording** menu.

6.5.6 Formatting templates

You can format an open template in the following ways (commands **Edit/Format Template** and **Options/Preferences/Format**):

- Modify the notation used for HTML and WTML tags
- Format the script source text
- Indent the HTML tags
- Indent the WTML tags

When you indent tags, any line breaks present in the original are preserved. However, you may specify that curly brackets and block tags are to be located on a separate line after formatting.

6.5.6.1 Modifying the notation used for HTML and WTML tags

The notation used for all HTML and WTML tags can be globally modified for a template.

There are the following two variants:

- All tags are written in uppercase, e.g. WTONCREATESCRIPT.
- All tags are written in lowercase, e.g. wtoncreatescript.

6.5.6.2 Formatting the script source text

The formatting of the script areas is subject to the following rules (option **Indent lines**):

- The indent level is incremented at the start of each block and is decremented again at the end of the block.
- Any continued statements are indented one level further than the start of the statement.
- Comments are indented to the same level as the surrounding text.
- Blank lines are reduced to just the line feed character.

Example

```
if ( wtCurrentComm_system.EDIT_MODE )
{
if ( typeof wtCurrentComm_system.isOverwrite == 'undefined'
&& wtCurrentComm_system.EDIT_MODE.match(/OVERWRITE/) )
wtCurrentComm_system.isOverwrite = true;
} else
wtCurrentComm_system.isOverwrite = false;
```

becomes:

```
if ( wtCurrentComm_system.EDIT_MODE )
{
   if ( typeof wtCurrentComm_system.isOverwrite == 'undefined'
      && wtCurrentComm_system.EDIT_MODE.match(/OVERWRITE/) )
      wtCurrentComm_system.isOverwrite = true;
} else
   wtCurrentComm_system.isOverwrite = false;
```

6.5.6.3 Indenting HTML tags

When an opening block tag is identified, the indentation level of the following lines is incremented and then decremented again when the end tag is identified. This does not affect the html tag.

In the case of a tag with an optional end tag, the indentation level is decremented as soon as a tag is recognized that implicitly closes this tag.

If a tag is longer than a line then all the lines through to the closing > are indented an extra level.

Inline tags and tags that have no end tag are treated as text.

Text within pre-tags is not modified.

For a subdivision of the HTML tags into classes that are of relevance for the calculation of the indentation level, see the WebLab online help system.

Unpaired tags

Due to the intermixture of various programming techniques, it is possible that HTML tags which by definition should be paired actually appear as unpaired tags.
Possible reasons for this include:

- A tag is opened in a script and is then closed in the HTML text or vice versa
- Source text has been exported to Include files.

In this case, it is possible that tags that belong together do not start in the same column.

Example

```
<html>
  <body>
    <script>
      document.write("<table border=1>");
    </script>
  </table>
</body>
</html>
```

Style tags

Within a style tag, the style formatting is enclosed in curly brackets. Consequently, indentation is performed within the curly brackets.

Example

```
<style>
    .h1
    {
        font-family:'Times New Roman', Times, serif;
        font-style:italic;
        font-weight:bold;
    }
</style>
```

6.5.6.4 Indenting WTML tags

In so far as possible, WTML tags are treated in the same way as HTML tags. You simply need to note the following comment for the tags for the control structures:

These tags do not modify the indentation level for HTML tags. However, the indentation level for further WTML tags for control structures is incremented. This guarantees that the opening and closing tags start in the same column even even if HTML and WTML tags are combined, for example if an HTML tag is opened inside an If tag and closed outside of it.

Example

```
<html>
<body>
<wtIf (a>b)>
    <table>
<wtElse>
    <table bgcolor="#ff00ff">
</wtIf>
        sometext
        <tr>
            <td>
            </td>
        </tr>
    </table>
</body>
</html>
```

If tags are opened within the If and the Else branch then only the tags of the If branch are used for the calculation of the indentation depth after the closing If tag.

6.5.7 Documenting templates

WebTransactions applications may contain many templates with complex functions and must be sufficiently documented. WebLab offers support when inserting comments in templates and the possibility to create, from the comments, a documentation in form of HTML pages. To do this, WebLab creates HTML pages from all the files in the base directory with a suffix `.htm`, `.html`, `.js`, `.clt`, and `.service`.

Only the following comments are considered during the generation of documentation:

- File comments, which are at the beginning of a file (see section “File comments” on page 184).
- Function comments which end in the line directly before the function definition (see section “Function comments” on page 184).

i To insert comments into templates or to generate HTML pages, you must have a connection to the WebTransactions server.

6.5.7.1 Format of the comments

All comments must be created according to the following rules. To help you create comments, WebLab provides a syntactically correct structure you only have to complete (see section “Inserting comments” on page 185).

- Every comment starts with the string `/**` .
- Every following line in a comment starts with an asterisk (`*`).
- The first sentence ends with the first period, occurring in the comment.
- The first sentence of the comment should briefly describe the function. This sentence is used as a short description in the function overview (see section “Example” on page 188).
- The description of the function ends with an empty comment line.
- The lines that contain symbols start after this empty comment line (for possible symbols see page 183).
- Lines containing a symbol must start with this symbol.
- Each comment ends with the string `*/` .

All HTML tags can be used for the formatting of comments. The text is taken over in the HTML pages unchanged.

Attention needs to be paid to the following points:

- Symbols which have a special meaning in HTML, must be coded (i.e. < or > as `&lt;` or `&gt;`).
- Line breaks in documentation must be created with HTML tools (i.e. with the tag `<br>`).
- The following symbols can be used in comments (see example on page 188). These symbols must be at the start of a line.

  `@author`
  indicates the author of the data. This symbol is only used in file comments.

  `@param`
  describes the parameter of a function. The following format needs to be maintained:

  `@param` *parameter description*

`@return`

describes the return value of the function.

`@throws`

describes the exception, the function throws. The following format needs to be maintained:

`@throws` *name_of_exception description*

`@method`

assigns, in a constructor comment, a function as method of a class. The following format needs to be maintained:

`@method` *method-name function-name*

File comments

These comments describe the content of a file. They must be at the beginning of the file.

File comments which are outside the script range must additionally be enclosed in `Rem` tags (exception: file comments in JavaScript files).

Example

```
<wtrem>
/**
 *
 * @author
*/
</wtrem>
<wtrem>
**********************************************************************
Functions for error handling
**********************************************************************
</wtrem>
```

Function comments

These comments are assigned to a specified function. Therefore they must end directly on the line above the function.

Example function comments can be found in section “Function comment” on page 185 and “Example” on page 188).

6.5.7.2 Inserting comments

WebLab has various functions which you use to enter comments into templates. You follow the following steps:

- ► Open the template that you want to edit.

File comment

- ► To add a file comment, in the context menu or in the **Add** menu, select the command **Insert Comment (File).**

  WebLab inserts a default for the comment at the file start.

Function comment

- ► To insert a function comment, position the cursor on the line in which the function is defined.
- ► In the context menu or in the **Add** menu, select the command **Function Comment**.

  This command is only available if the cursor is in the line in which the function is defined.

  WebLab inserts a default for the function comment immediately above the function. WebLab analyses the function call and inserts the appropriate lines with the symbol `@param` for all parameters.

Examples

WebLab inserts, depending on the function call, the following comment:

Function call

```
function test (a,b,c)
```

Comment

```
/**
 *
 *
 * @param a
 * @param b
 * @param c
 * @return
 * @throws
*/
```

Function call

```
function WT_SOAP( /*string*/ wsdlSrc, /*string*/ proxyHost, /*number*/
proxyPort )
```

Comment

```
/**
 *
 *
 * @param wsdlSrc [string]
 * @param proxyHost [string]
 * @param proxyPort [number]
 * @return
 * @throws
*/
```

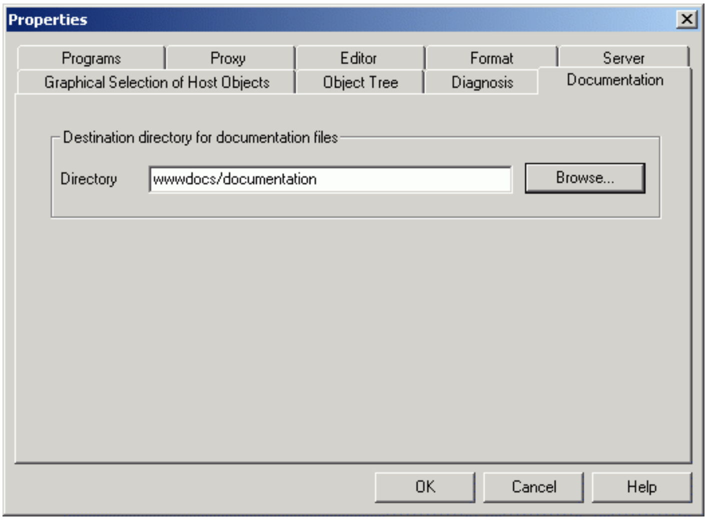
6.5.7.3 Generating documentation

To create HTML pages from your comments, use WebLab and proceed as follows:

As the first step, determine where the HTML pages should be created.

- ► Choose the command **Options**/**Preferences**.
- ► In the dialog field **Preferences** choose the tab **Documentation**.
- ► Indicate in the field **Directory**, where the HTML pages should be created.
  The default for this value is *wwwdocs/documentation*.

  The target directory of the documentation must be available via WebLab, as well as via the Web server. The directory must be:
  - – under the directory `wwwdocs` on the WebTransactions server or
  - – on the local computer under the root directory for Web pages (= document directory).



Directory: wwwdocs/documentation

Subsequently the HTML pages can be created:

- ► Choose the command **Generate**/**Documentation**.

  WebLab creates HTML pages from all files in the base directory with the suffix `.htm`, `.html`, `.js`, `.clt` and `.service`. Links to the installation directory are not taken into consideration.

  The HTML pages are put in the directory indicated in the **Preferences** dialog box. They can be opened with any browser.

You can call the documentation from WebLab:

- ► Choose the command **?**/**Display Documentation**.

  WebLab opens a browser window with the start page of the documentation

6.5.7.4 Format of the display

You can display the documentation of a WebTransactions application with the command **?**/**Display Documentation**.

There are two display possibilities for the HTML pages. You choose the display through the button **Files** or **Index** at the header of each page.

File display

If the button **Files** is chosen, an overview of the functions is displayed in sequence of the order they appear in the file. The HTML page contains in this sequence:

- the file comment.
- an overview of the functions in the file, sorted by server sided and client sided defined functions. The overview contains for each function the definition and the first sentence from the function comment. The detail view of the function is available through a Link.
- Detail view of all functions. These contain the complete function comment.

Alphabetic display

To display an alphabetical list of all the functions in the file, select the **Index** button. For each function the list gives the definition and the first sentence of the function comment. The detail view of the function is available through a link.

6.5.7.5 Example

The file `test.htm` contains the function `WT_SOAP` together with the following comments:

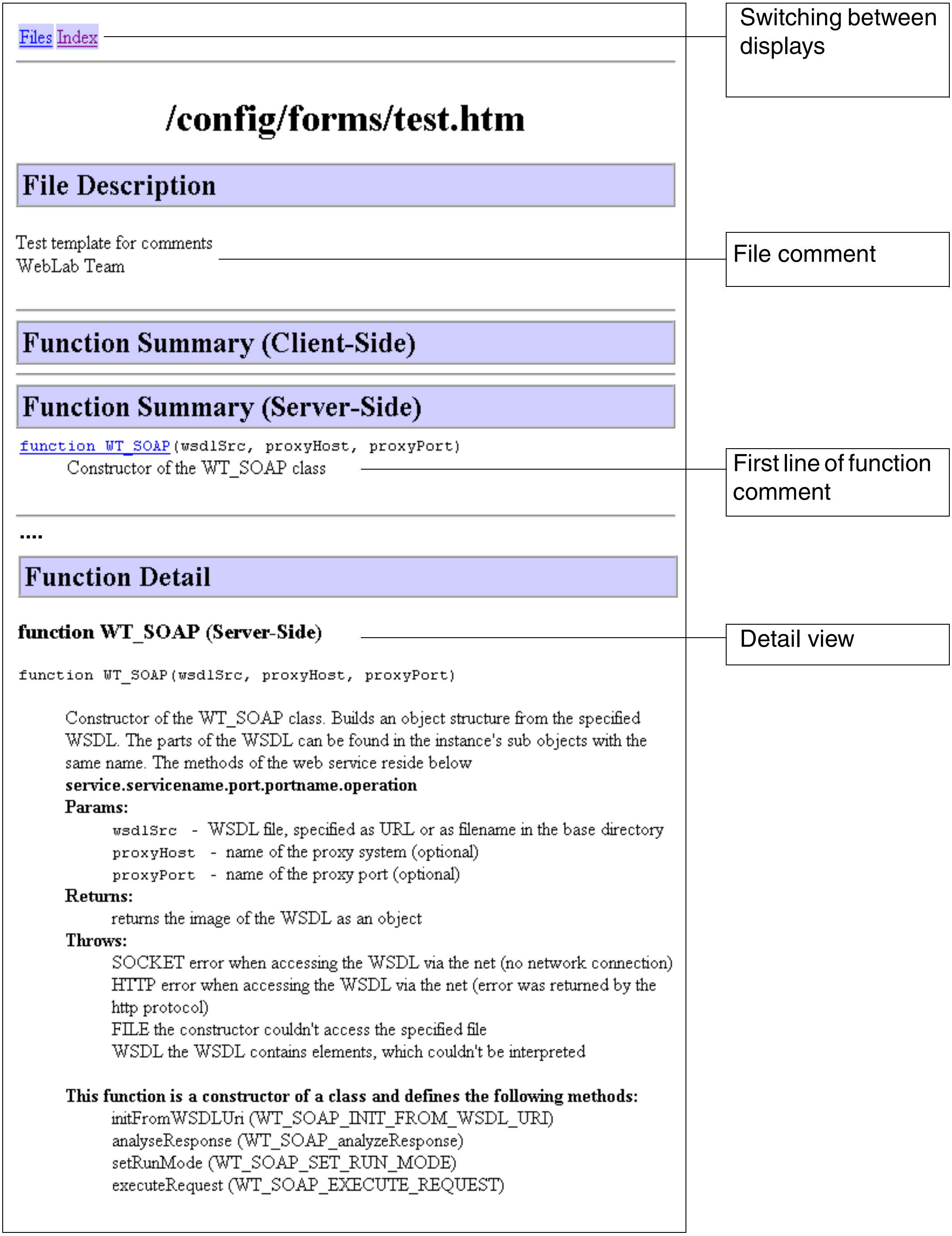
```
<wtrem>
/**
 *
 * Test template for comments <br>
 * WebLab Team
*/
</wtrem>
<wtoncreatescript>

/**
*  Constructor of the WT_SOAP class.
 * Builds an object structure from the specified WSDL.
 * The parts of the WSDL can be found in the instance's
 * sub objects with the same name.
 * The methods of the web service reside below
 *  <b>service.servicename.port.portname.operation</b>
 *
 * @param wsdlSrc WSDL file, specified as URL or as
 * filename in the base directory
 * @param proxyHost name of the proxy system (optional)
 * @param proxyPort name of the proxy port (optional)
 * @return returns the image of the WSDL as an object
 * @throws SOCKET:   error when accessing the WSDL via the net
                    (no network
connection)                                          (1)
 * @throws HTTP:     error when accessing the WSDL via the net
                    (error was returned by the http
protocol)                        (1)
 * @throws FILE:     the constructor couldn't access the specified
file              (1)
 * @throws WSDL:     the WSDL contains elements, which couldn't
                     be
interpreted                                                  (1)
 * @method initFromWSDLUri
WT_SOAP_INIT_FROM_WSDL_URI                               (2)
 * @method analyseResponse
WT_SOAP_analyzeResponse                                   (2)
 * @method setRunMode
WT_SOAP_SET_RUN_MODE                                         (2)
 * @method executeRequest
WT_SOAP_EXECUTE_REQUEST                                    (2)
*/
function WT_SOAP(  wsdlSrc, proxyHost, proxyPort )
....
```

```
....
```

(1) The class `WT_SOAP` has several exceptions, which differ from each other by name.

(2) In the class `WT_SOAP` these functions are available as methods.

In the created documentation the following HTML page is obtained:



Files Index

/config/forms/test.htm

File Description

Test template for comments
WebLab Team

Function Summary (Client-Side)

Function Summary (Server-Side)

`function WT_SOAP(wsdlSrc, proxyHost, proxyPort)`
Constructor of the WT_SOAP class

....

Function Detail

function WT_SOAP (Server-Side)

```
function WT_SOAP(wsdlSrc, proxyHost, proxyPort)
```

Constructor of the WT_SOAP class. Builds an object structure from the specified WSDL. The parts of the WSDL can be found in the instance's sub objects with the same name. The methods of the web service reside below **service.servicename.port.portname.operation**

Params:
`wsdlSrc` - WSDL file, specified as URL or as filename in the base directory
`proxyHost` - name of the proxy system (optional)
`proxyPort` - name of the proxy port (optional)

Returns:
returns the image of the WSDL as an object

Throws:
SOCKET error when accessing the WSDL via the net (no network connection)
HTTP error when accessing the WSDL via the net (error was returned by the http protocol)
FILE the constructor couldn't access the specified file
WSDL the WSDL contains elements, which couldn't be interpreted

This function is a constructor of a class and defines the following methods:
initFromWSDLUri (WT_SOAP_INIT_FROM_WSDL_URI)
analyseResponse (WT_SOAP_analyzeResponse)
setRunMode (WT_SOAP_SET_RUN_MODE)
executeRequest (WT_SOAP_EXECUTE_REQUEST)

6.6 Testing templates

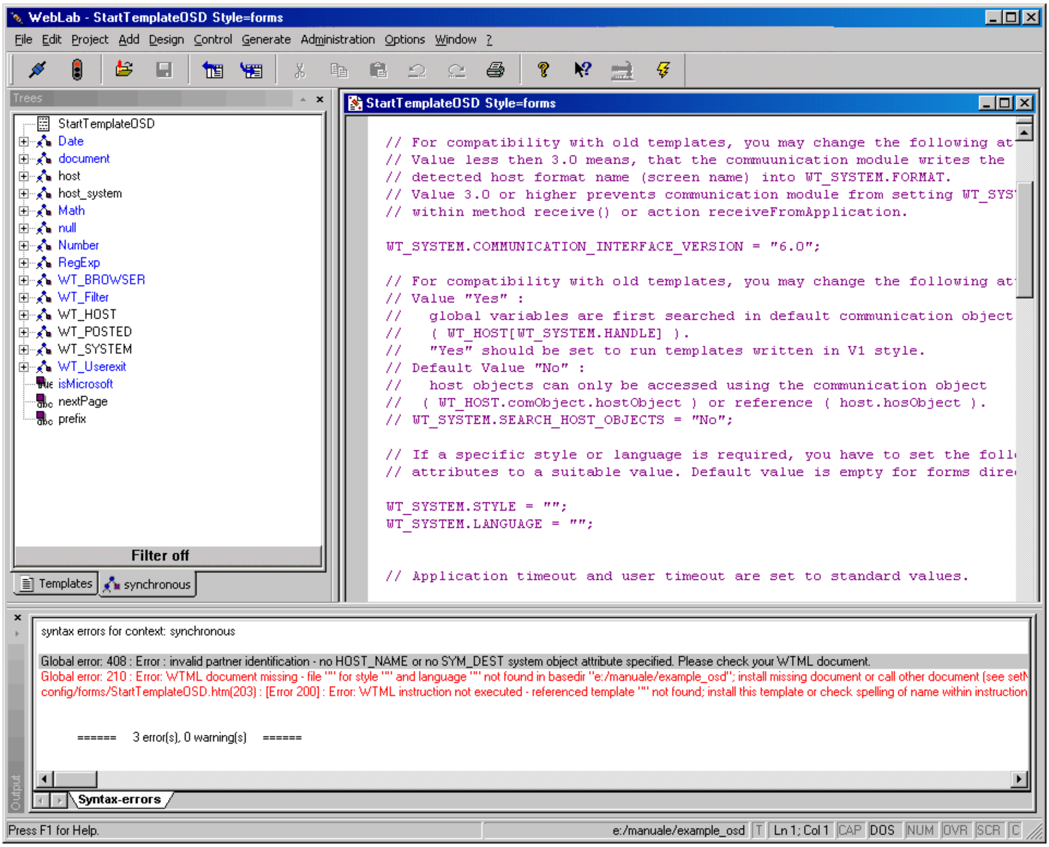
There are various ways of testing your WebTransactions applications:

- Testing the design of a template
- Testing the execution sequence of a template (single step tracking)

i The test functions of the WebLab development environment are outlined below. For a detailed description, refer to the WebLab online help.

6.6.1 Testing the design of a template

WebLab provides you with the following user-friendly ways of testing the design of an HTML page.



```
// For compatibility with old templates, you may change the following at
// Value less then 3.0 means, that the commuunication module writes the
// detected host format name (screen name) into WT_SYSTEM.FORMAT.
// Value 3.0 or higher prevents communication module from setting WT_SYS
// within method receive() or action receiveFromApplication.

WT_SYSTEM.COMMUNICATION_INTERFACE_VERSION = "6.0";

// For compatibility with old templates, you may change the following at
// Value "Yes" :
//   global variables are first searched in default communication object
//   ( WT_HOST[WT_SYSTEM.HANDLE] ).
//   "Yes" should be set to run templates written in V1 style.
// Default Value "No" :
//   host objects can only be accessed using the communication object
//  ( WT_HOST.comObject.hostObject ) or reference ( host.hosObject ).
// WT_SYSTEM.SEARCH_HOST_OBJECTS = "No";

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

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


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

```
syntax errors for context: synchronous

Global error: 408 : Error : invalid partner identification - no HOST_NAME or no SYM_DEST system object attribute specified. Please check your WTML document.
Global error: 210 : Error: WTML document missing - file "" for style "" and language "" not found in basedir "e:/manuale/example_osd"; install missing document or call other document (see setN
config/forms/StartTemplateOSD.htm(203) : [Error 200] : Error: WTML instruction not executed - referenced template "" not found; install this template or check spelling of name within instruction


    ======    3 error(s), 0 warning(s)    ======
```

Press F1 for Help. e:/manuale/example_osd T Ln 1; Col 1 CAP DOS NUM OVR SCR C

Correcting errors

If WebTransactions issues error messages then these are displayed in the WebLab output area. If you double-click on the error message, the cursor is automatically positioned at the location of the error in the edit area. You are then able to correct it immediately.

You can use the **Control/Update In Browser** command to regenerate the modified page immediately and display it in the browser.

Setting new values

During the test phase, it is often useful to set new values for certain attributes and check the effect this has. To do this, select the required object in the object tree.

Using the command **Properties** in the context menu, the current value is shown and can be edited. Like with error corrections, you can immediately verify the result by using the command **Update in Browser** to create and view the page with the modified values.

Notes on the “Update in Browser” function

You use the **Control/Update in Browser** command to output a page again following a correction in the template and check your modification.

i However, when using this function you should be aware of certain limitations which can make the result of regeneration differ from normal output:

- System and host objects are not reset to their original status: if attributes of these objects are modified in OnCreateScript tags then the regeneration is based on the modified values.
- Communication steps with the host application are not reset: if the methods `open`, `close`, `send` and `receive` are used, their effect cannot be undone. The communication steps are suppressed on regeneration in order to leave the host application in its current state. Changes to these OnCreateScripts (add, delete) therefore have no direct effect. Regeneration may therefore lead to an inconsistent host application state.

  In contrast, OnReceiveScripts are completely uncritical since regeneration prevents their execution. I.e. the old statements stored for Receive time are discarded and the new OnReceive tags or OnReceive scripts are saved (see also section “Dialog cycle” on page 62).

  In rare cases, problems may arise on regeneration. In such cases, you must navigate back to the test point in order to test the associated template. Normally, for example in the case of all generated templates, regeneration is performed without problems and considerably accelerates the user interface test.

6.6.2 Testing the execution sequence in the template

You can track the execution sequence of a template by first logging all the steps that are performed in the template during the generation of the HTML pages and the processing Receive scripts. When you do this, WebTransactions writes a logfile using the following rules.

- The variables and their associated values are logged after every executed line.
- The class templates of host objects are also executed and the results are logged.
- The generation of an HTML area is a single step. It is indicated by the display of the last line in the HTML area.
- If multiple WebTransactions steps are executed within a single dialog step (e.g. in the case of framesets), then only the single steps from the last generation are displayed.

WebLab then uses this logfile for single step tracking.

i Please note that WebTransactions normally deletes the log file when the session is deleted. If you wish to log single steps including the end of the session, you must also activate the WebTransactions Trace (see section “Trace functions” on page 121).

You can use single step tracking to

- follow execution through a template
- track execution within a specific area of a template

The current variables and their values are displayed in the **Single Step Tracking** tab in the output area.

6.6.2.1 Tracking execution via a template or a template area

► You can choose between the following two ways of starting to track execution:

  ► To track the individual steps in a complete template, you activate logging with the command **Control/Single Step Tracking/Record Single Steps**.

  or:

  ► To test execution only within a given area in the template, you activate logging for the area directly in the template with the WTScript function setSingleStep(“on“|“off“). For more information, see the WebTransactions manual “Template Language” and the example below.

  The subsequent steps apply equally to both alternatives.

► Run the session using the template that you want to test.

- ► Start single step tracking with the command **Control/Single Step Tracking/ Begin Single Step Tracking**.
- ► Use the commands in the submenu **Control/Single Step Tracking** or the icon bar **Single Step Tracking** to trace the execution of the program. WebLab highlights the current statement in the template and displays the variables used by this statement together with the associated values in the output area.

➪ For a detailed description of the commands in the **Single Step Tracking**, see the detailed WebLab online help system.

Example

In this example, execution is simply tracked for an area of a template:

```
...
<wtOnCreateScript>
<!--
  wtInputFields = new Object;
  currentLine   = 1;
  for (element = OSD_O.$FIRST.Name; OSD_O && element != '$END'; element =
OSD_O.$NEXT.Name)
  {
    currentHostObject = OSD_O[element];
    if ( currentHostObject.StartLine != currentLine )
    {
      document.writeln();
      currentLine++;
    }
    if ( currentHostObject.Type == 'Protected' &&
currentHostObject.Markable == 'No' )
    {
      setSingleStep ("on");
      taggedOutput( currentHostObject );
      setSingleStep ("off");
    }
    else
    {
      wtInputFields[ element ] = currentHostObject;
      taggedInput( currentHostObject );
    }
  }
//-->
</wtOnCreateScript>
...
```

6.6.2.2 Monitoring the values of variables

You can use the value window to monitor the values of variables, see also section “The value window” on page 168.

➪ For detailed information on working with the value window, see the detailed WebLab online help system.

6.7 Integrating server tools in WebLab

As the WebTransactions administrator (user ID `admin`) you can make commands that execute on a WebTransactions server available to users in WebLab. The term “tools” covers both commands and programs.

i Please note that the commands must be batch-compatible. This means that the commands must not require any user input.

You define tools using the command **Options/Edit Server Tools**.

When you formulate a command, you can use certain keywords for the parameters. When you execute a WebLab command, these keywords are replaced by the values for the current session.

Keyword	Replaced by:
`%BASEDIR`	Current base directory
`%CURRENT`	Name of the file in the current editing window including path e.g. basedirs\test\config/forms/trav0.htm
`%CURRENTNAME`	Name of the active file, e.g. trav0.htm
`%CURRENTPATH`	Path of the active file (Windows: without drive specification e.g. \basedirs\test\config\forms)
%DRIVE	Drive for the current file (Windows only)
`%SESSIONID`	ID of the current session
%SELTEXT	Text selected in the current window
`%USER`	Active WebTransactions user
`%1,%2,...,%9`	Parameters that all users can specify themselves using **Options/Customize Server Tools**.

You can also define the users who are authorized to work with the tools.When these users connect to a base directory on the server, they see the tools that are available to them in the submenu **Control/Server Tools**. Users can then click on a tool to run it on the server.

Click the tool to run it on the server. When a tool is processed, all the other tools are locked. If you terminate the connection, operation of the tool will also be terminated.

Outputs from the tools are displayed on two tabs in the output area:

Server tools(stdout)

Contains outputs that the tool sends to the console or the screen.

If a line of the output contains a file name, double click this line to open the file displayed. If the line also includes a line number, the cursor will be moved to the line indicated by the number. The lines, which can be processed this way, have the following format:

filename:*linenumber*:*text*
filename(*linenumber*) *text*
filename:*text*

Server tools (stderr)

Contains any tool error messages.

6.8 Transferring and distributing a WebTransactions application

You transfer a WebTransactions application by packing it at the development computer and then unpacking it at the destination machine. On distribution, the files or directories are read into WebLab and are then transferred to the destination hosts in a cluster.

The precise procedure is described below.

6.8.1 Scope of transfer

Before packing the data in an archive, you can specify which directories and files you want to transfer. The table below summarizes how the files in the various subdirectories of the base directory are transferred.

Directory	**Contents**	**Transfer mode**
`basedir`	Program files, libraries	Binary
`config`	FLD files	Text
`forms`	Templates	Text
`msg`	Message files	Text
`tmp`	Temporary files	Not transferred
`wwwdocs`	Documents for the web server	Text
`applet`	Applets	Binary
`class`	Java classes	Binary
`html`	HTML pages	Text
`image`	Images	Binary
`javascript`	Client-side JavaScripts	Text
`style`	Stylesheet definitions	Text
own directory		Text

Table 2: Transfer modes for the data in a base directory

When you unpack the data at the target computer, you can decide whether you want to create a new WebTransactions application or whether you want to extend an existing WebTransactions application. Once an WebTransactions application has been transferred, the paths of the WebTransactions CGI programs are adapted in all the files in the `wwwdocs/html` subdirectory if necessary.

ℹ Please note that when you unpack and you are connected to a WebTransactions application, any open, removed files are not automatically reloaded. However, WebLab informs you that there is a new version of the file on the server and loads this new version on request.

6.8.2 Packing an application

You pack an application using the command **Administration/Pack Application**.

During the packing operation, you have two ways of defining which files are to be packed:

- You can activate a filter to hide files from the displayed tree structure. You can filter on filename suffix and/or creation date, see also "File selection" on page 166.

  Files that are not displayed are also not packed.

- In the displayed tree structure, you can also explicitly include or exclude files. You do this by clicking on the icon located next to each entry.

All the selected files or directories are packed in the WebTransactions archive. In the WebLab output window, you will see messages informing you of the progress of the pack operation in the **Pack&Go** tab.

6.8.3 Unpacking an application

To unpack an archive that contains a WebTransactions application, use the command **Administration/Unpack Application**.

If you are connected to a base directory when you call this command, the application is unpacked to this base directory.

If you are not connected to a base directory then a base directory is created and the application is unpacked to this directory.

If the virtual paths to `WTPublish.exe`, `WTPublishISAPI.dll` or `WTCluster.exe` are different at the source and target hosts, you can adapt these paths during the unpacking operation.

In the WebLab output window, you will see messages informing you of the progress of the unpack operation in the **Pack&Go** tab.

6.8.4 Distributing an application

Individual files or directories are read into WebLab before being transferred to the target host in the cluster.

If you want to distribute multiple files or directories, choose the command **Administration/Distribute Application**.

In the same way as when you transfer an application (see section “Packing an application” on page 199 and section “Unpacking an application” on page 199), these files are packed in an archive file for distribution and this file is then unpacked at all the hosts in the cluster.

You have the same possibilities of defining the scope of the archive as when packing an application (see section “Packing an application” on page 199). For example, you can use a filter to distribute only those files that have been modified after a certain date, see page 166.

7 Appendix: demo applications

When you install WebTransactions, you can also install a number of demo applications. These are stored in the web server's document directory.

You start the demo applications with `http://`*machine*`/webtav75/startdemos.htm`. This displays a start page on which you can choose between the different demo applications. Once you have chosen a demo application, a further page explains the operation of the application. The templates are accompanied by detailed comments.

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

(**C**ommon **G**ateway **I**nterface)
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 **H**igh-**L**evel **L**anguage **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

(**E**nterprise **J**ava**B**ean)
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 **H**andling **S**ystem
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

(**H**yper**t**ext **M**arkup **L**anguage)
See *->Hypertext Markup Language*

HTTP

(**H**ypertext **T**ransfer **P**rotocol)
This is the protocol used to transfer *->HTML* pages and data.

HTTPS

(**H**ypertext **T**ransfer **P**rotocol **S**ecure)
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

(**H**yper**t**ext **M**arkup **L**anguage)
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

(**L**ightweight **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

(**M**aster **T**emplate 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

(**U**niversal **T**ransaction **M**onitor)
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 ->Web*Transactions 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

(**T**ransport **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 permity 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

(**W**eb **S**ervice **D**efinition **L**anguage)
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

(**W**eb**T**ransactions **M**arkup **L**anguage)
Markup and programming language for WebTransactions *->templates*. WTML uses additional ->*WTML tags* to extend *->HTML* and the server programming language ->*WTScript*, e.g. for data exchange with *->host applications*. WTML tags are executed by WebTransactions and not by the *->browser* (serverside scripting).

WTML tag

(**W**eb**T**ransactions **M**arkup **L**anguage-Tag)
Special WebTransactions tags for the generation of the dynamic sections of an *->HTML* page using data from the*->host application*.

WTScript

Serverside programming language of WebTransactions. WTScripts are similiar 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

(e**X**tensible **M**arkup **L**anguage)
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 (**D**ocument **T**ype **D**efinition), XML Schema (W3C standard) or XDR (**X**ML **D**ata **R**educed).

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**yper**t**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**t**ext **T**ransfer **P**rotocol **S**ecure
IFG	**I**nteraktiver **F**ormat **G**enerator
ISAPI	**I**nternet **S**erver **A**pplication **P**rogramming **I**nterface
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
LPD	**L**ine **P**rinter **D**aemon
MT-Tag	**M**aster-**T**emplate-Tag
MVS	**M**ultiple **V**irtual **S**torage
OSD	**O**pen **S**ystems **D**irection
SGML	**S**tandard **G**eneralized **M**arkup **L**anguage
SOAP	**S**imple **O**bject **A**ccess **P**rotocol

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

Related publications

WebTransactions manuals

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

WebTransactions
Template Language
Reference Manual

WebTransactions
Client APIs for WebTransactions
User Guide

WebTransactions
Connection to openUTM Applications via UPIC
User Guide

WebTransactions
Connection to OSD Applications
User Guide

WebTransactions
Connection to MVS Applications
User Guide

WebTransactions
Access to Dynamic Web Contents
User Guide

WebTransactions
Web Frontend for Web Services
User Guide

Index