



**PRIMEBASE
APPLICATION SERVER
USER'S GUIDE**

July 2008

PrimeBase Systems GmbH

Max-Brauer-Allee 50 - D - 22765 Hamburg - Germany

www.primebase.com - e-mail: info@primebase.com

1. Introduction	1
2. Key Concepts	1
Web Applications	1
User Agents and Browsers	2
Web Program Pages	2
Template Documents	2
Program Modules	2
PrimeBase Enterprise Objects	2
Browser Identification	3
Proxies and Caching	4
The PrimeBase Virtual Machine	4
Programming the PBVM: PBT	4
Sessions and Clones	4
Connections & Databases	5
The Item Stream Interface	5
Escape Commands	5
Processing Page Requests	6
The Operation Cycle	6
Browser and Pool Sessions	6
Page Compilation and Caching	7
Page Execution	7
3. Installing & Running	7
Installing the Application Server	8
Starting the Application Server	9
Activating the Application Server	9
Accessing the Application Server from a Browser	10
Accessing the Application Server via a Web Server	11
Accessing the Application Server via a Web Server from a Browser	13
4. Configuring the Application Server	14
Setting System Parameters	14
Running Multiple Instances of the Application Server	15
Control/Mapping Files	16
5. System Web Interface	16
System Module Overview	17
The Application Tab	18
Login	18
Online Manuals	18
Example Page	18
Directory Browser	18
PBE Demo	18
Restart	18
Logout	18
The Layout Tab	19
The Programmer Tab	19
The Owner Tab	19
The Configuration Tab	20
Deployment	20
Connections	20
Parameters	22
Reconnect	23
6. Directories and Files	23
Main Directory	23
PrimeBase Application Server Executable	23
Start Server Shell Scripts	23
Stop Server Shell Script	23

Server Console Shell Script	24
Service Executable	24
Stop Server Executable	24
PrimeBase Console	24
PrimeBase Application Server Engine	24
PrimeBase Virtual Machine	24
PrimeBase SQL Database Server	25
Environment Editor	25
Automation Client	25
Connection Setup	25
Shared Memory Manager	25
Registration Key	25
Read Me	25
Release Notes	26
Documentation	26
Setwrite Tool	26
WebSTAR Plug-in	26
'Cgi-bin' Folder	26
'Docs' Folder	26
'Setup' Folder	26
'Cgi-bin' Folder	27
PrimeBase CGI Application	27
Apache Server Plug-in	27
ISAPI Interface	27
'Docs' Folder	27
Default Source Document	27
PBE Demo	27
Directory Browser	28
Example Page	28
PrimeBase Manuals	28
System Module	28
PrimeBase Tutorial Demo	28
'Setup' Folder	28
Connection Definition	28
Action Map	28
Livetag Map	29
PBAS System Parameters	29
PBVM Parameters	29
Input Character Conversion	29
Output Character Conversion	29
Registered Users	29
Protocol	29
'Exec' Folder	30
'Initialize' Folder	30
'Startup' Folder	30
'Unicode' Folder	30
'Custom' Folder	30
Connection Definition	30
Character Conversion	31
7. Application Server Actions	32
Action Maps	32
Types of Actions	33
EXECUTE	33
RUN	33
RETURN	33
NOACCESS	34
Action Options	34
PREPAGE	34
INPUT	35
PREGEN	35
HEADER	36

LIVETAGS	36
Execution Options	36
Session Type	37
Page Type	37
Proxy Caching	38
URL Browser ID	38
8. LiveTags	39
What are LiveTags?	39
Defining LiveTags	39
Tag Specification	40
LiveTag Options	40
Handler Functions	41
Handler Function Parameters	41
Parameter Options	41
Parameter Scope	41
Parameter Types	42
PBT Attribute Type	43
Looping and Branching	43
LiveTag Control Blocks	43
LiveTag Escape Commands	43
Appendix	44
A. Escape Commands	44
Session / Execution Control	44
Set Time-outs Locally	44
Errors, Debugs, Protocols and Comments	45
Binary Value Control	45
Set/Get/Callback Variable Escapes	46
LiveTag Escapes	47
Background Processing	48
B. System Parameters	48
General Configuration	48
Session Creation & Configuration	49
Session Time-outs	49
Protocol Logging / Display	50
Identification of Browsers	50
BLOB Files & URL Parameters	51
List Variable Pattern	52
Activation Parameters	52
HTTP Controls	52
Error Page Settings	53
C. Application Server Errors	53

PRIMEBASE APPLICATION SERVER

USER'S GUIDE

The PrimeBase Application Server is part of a development platform for the production of various kinds of applications, for example client/server software and Web applications. This platform consists of two main products: the PrimeBase SQL Database Server (PBDS) and the PrimeBase Application Server (PBAS), both of which are based on open standards.

The major task of the PrimeBase Application Server is to create a connection between the Web and the PrimeBase Virtual Machine (PBVM). The PBVM in turn functions as the "control hub" of the entire system, implementing program logic and transferring data to and from diverse data sources. In particular, PBAS connects an internet browser, in general an internet "user agent", to a session. A session is a protected application runtime environment created by the PBVM.

The so called PrimeBase Open Server Interface (PBOS) provides native interfaces to a large number of 3rd party DBMSs and other data sources. In this way, the application server can integrate and publish data from legacy systems and complement existing infrastructure.

This document describes the features of the PrimeBase Application Server and is intended for both system administrators and programmers. We discuss how PBAS works and how to tune it for special demands and purposes. A look at development and programming aspects is done only as far as it is necessary to understand the basic functioning of PBAS. How to create a Web application using the PrimeBase Application Server is described in the "PrimeBase Enterprise Objects Programmer's Manual".

Tip

For more information about the entire PrimeBase System please refer to the "PrimeBase System Overview".



In this chapter we present an overview of the operation of the PrimeBase Application Server. Understanding of the basic concepts explained here is a prerequisite for the discussion of various aspects of the application server in the following chapters.

A Web application is an application based on internet standards. This includes standards such as:

- TCP/IP: Transport Control Protocol/Internet Protocol. The basic end-to-end communications protocol of the internet.
- HTTP: Hypertext Transport Protocol. A protocol based on TCP/IP originally designed as the communications protocol between internet user agents (see below) and Web servers.
- HTML: Hypertext Markup Language. A tag based text format used to display graphic and text and describe user interfaces.
- XML: Extensible Markup Language. An extendable, tag based text format used to describe the structure and value of data.
- WAP: Wireless Application Protocol. A protocol designed specifically for wireless devices such as mobile telephones.

An application that uses these protocols is equipped to work in an Internet, Intranet or Extranet environment.

1. Introduction

2. Key Concepts

Web Applications

Web applications are also known as "server-side" applications. This means that multiple instances of the program run in a central location remote from the user-interface.

User Agents and Browsers

A user agent is the client-side component of a Web application. As a result, the user interface aspects of the program are handled by the user agent. In this case the user agent is known as a browser. Most people are familiar with PC based browsers, but browsers are also available in devices such as PDAs (Personal Digital Assistants), mobile telephones and televisions set-top boxes.

User agents also include programs that gather information such as the robots that scan the Web for data to be indexed for searching purposes and programs that enable offline browsing.

For the purpose of e-Commerce and IT infrastructure in general, user agents designed for the integration and exchange of data will become even more important. In this case we have the situation in which the application server functions as a user agent itself. The PrimeBase Application Server offers features that enable it to fulfill this requirement.

Since browsers are the most common type of user agent this document often just refers to browsers when, in fact, any type of user agent is implied.

Web Program Pages

A Web program consists of program code and source pages. The code implements the program logic, while the pages describe the user interface of the application.

Generally speaking, the end-user of the Web application, using a Web browser, initiates an exchange with the application server by requesting a particular source page of the program. The application server processes the request and returns a result page to the browser. The browser then displays the page for the user.

Note that a source page may not necessarily be associated with a particular file on disk. This is because it is possible to generate a result page entirely using program code and, for example, data from a database. PBAS supports all kinds of page generation, including code-based generation, and so-called "template-based" generation.

Template Documents

If a source page has an associated disk file, then this is known as a template document and is located in the application server's 'Docs' folder (see page 27).

Template documents consist of both static and dynamic components. The dynamic components of a PBAS template document may either be embedded code or "LiveTags". LiveTags™ is a technique by which standard HTML or XML tags are filled or replaced with dynamic content (see chapter "LiveTags" on page 39).

The dynamic components are executed and the output, combined with the static components of the document, form a result page.

Program Modules

A Web application developed with for the PrimeBase Application Server is divided into a number of independent "modules". Each module has its own code and template documents. All the files belonging to a module are located in sub-directories of the application server's 'Docs' folder. Modules can be installed or removed simply by moving the sub-directory and its contents in or out of the 'Docs' folder.

PrimeBase Enterprise Objects

PrimeBase Enterprise Objects (PBEO) is an object-oriented framework that provides the basic components needed to build a Web application. The PBEO frame-

work is pre-installed and shipped with the application server.

PrimeBase Enterprise Objects are programmed by defining the objects that make up the application. By extending standard PBE classes you can implement business logic particular to your application.

LiveTags are provided by the framework to display the information in an object in the browser. PBE also provides methods for doing all standard database operations.

These features and others make creating a Web application with the PrimeBase Application Server extremely efficient. For getting started with PrimeBase Enterprise Objects we recommend the "PrimeBase Tutorial". Detailed information is provided by the "PrimeBase Enterprise Objects Programmer's Manual".

Browser Identification

An essential part of the functioning of a Web-based application is the identification of the user agent or browser. As a result, every browser that interacts with the application server is given a unique identification number or ID. The application server requires that the browser include the ID with each page request it makes. If a request arrives without an ID, then the application server considers this to be a "new comer" and issues the browser a new ID.

There are two ways in which the ID can be transported to the application server.

HTTP Cookies

HTTP cookies is a mechanism provided by HTTP which allows a server-side program to store a named value in the browser. This value is called a cookie. For security reasons cookies may only be returned to the server that sent them.

Permanent cookies are stored by the browser on the client machines hard drive. Such cookies have an expiry date that may be set by the server.

Temporary cookies are stored in RAM and are only valid as long as the browser program is running.

PBAS only makes use of temporary HTTP cookies - if at all.

HTTP cookies may have been disabled by the user, or not supported by the browser. In this case it is necessary to use URL IDs in order to identify the user agent.

URL ID

A Universal Resource Locator (or URL) is the absolute address of a resource on the internet. The browser uses a URL to identify and request a particular source page of the Web application. Name/value pairs, known as search args, may be appended to a URL.

The application server uses this mechanism to attach the browser ID to the page request. The browser ID is added as a search arg to all URLs in result pages returned by the application server. In this way, subsequent requests that use URLs placed in previous result pages include the browser ID.

Two problems must be considered when using URL IDs. Firstly, if the user leaves the Web application for some other Web site URL IDs are lost. This means the user is considered a "new comer" when he returns to the Web application. This is not the case with HTTP cookies (that is, as long as the user does not close the Web browser).

Secondly, an ID in a URL can become "stale". This means that the application server has discarded its information associated with the ID. How long the application server maintains this information depends on time-out values set on the server. A request with a stale ID causes a new ID to be issued to the browser, and most often an error message is returned to the user. The programmer should then ensure that all URLs with stale IDs are removed from the browser to prevent loop-

ing. This can be done by ensuring that the entire browser window is reloaded.

Proxies and Caching

Caching by both internet proxies and browsers is used to shorten load times and reduce traffic on the internet. As long as the source pages are static this is a useful feature. Dynamic result pages generated by a Web application, however, should not be cached.

To prevent proxies and browsers from caching pages the application server uses two methods. The first method uses the "PRAGMA: NOCACHE" HTTP header option to specify that a page should not be cached. The second method is based on the fact that proxies and browsers use the URL of a source page as an index to locate a page in cache. By slightly modifying all URLs returned to the browser the application server prevents the result pages from being cached.

The PrimeBase Virtual Machine

As mentioned above the PrimeBase Application Server, in conjunction with the PrimeBase Virtual Machine runs a Web application. In this section we describe some aspects of the PBVM that are relevant to the PrimeBase Application Server. General information on the PBVM is available in the "PrimeBase System Overview".

Programming the PBVM: PBT

The PBVM is the execution unit of the PrimeBase System. It loads, compiles, optimizes and executes code written in PrimeBaseTalk (PBT).

PBT is an object-oriented programming language that combines the standard database language SQL (Structured Query Language) with a Java™-like syntax and program constructs. If a Java™ Virtual Machine is available then the PBVM allows seamless integration of Java™ and PBT objects.

PBT is easy to learn and is ideal for the implementation of business logic for several reasons, for example:

- PBT has native support for database types such as VARCHAR, DECIMAL, BLOBs and NULL values.
- PBT implements database cursors including operations such as FETCH FIRST, LAST, NEXT, PREVIOUS and FETCH ABSOLUTE.
- PBT has built-in support for transactions, including the statement BEGIN, COMMIT and ROLLBACK.

A built-in compiler in the PBVM makes it possible to ship PBT programs in source code form. The PBVM is also able to output programs as encrypted byte-code when applications and libraries are to be shipped for commercial purposes.

Sessions and Clones

The PrimeBase Virtual Machine creates a runtime environment called a session. Each session has its own address space and execution thread. The program running in a session is known as an application instance.

Application instances run independently of one another. The session boundaries protect application instances from failure and memory overwrites of other instances.

All sessions share the same initialization code. This is due to the fact that all sessions are created by duplication of an original "clone session". The clone session, in turn, loads the initialization code. The PBVM system file, 'initialize.sys', which is loaded and executed by the clone session, is responsible for loading the initialization code.

After a session has been created, the system file 'startup.sys' is loaded and executed. This file loads the start-up code of the session.

The clone concept allows the PBVM to create a new application instance extremely fast. This is due to the fact that when the clone session is duplicated, not just code, but all program data, connections and settings are duplicated as well. The new application instance is therefore ready to run immediately.

Connections & Databases

Each session may have a number of connections to various database servers or other data sources. Connections to data sources other than the PrimeBase SQL Database Server are possible using the PrimeBase Open Server Interface. For example, the PrimeBase Open Server provides native support for Oracle™ and DBMSs supporting ODBC.

Each connection can have a number of open databases, as far as the database concept is supported by the data source.

The OPEN DBMS and OPEN DATABASE statements make it possible to open connections and databases and close them, using the CLOSE statement, at any time.

To open a connection a "connection definition" is required. A connection definition specifies information such as the protocol to be used for the connection, the host machine on which the server is running, and the name of the server. A connection definition can also specify a default database to be opened, and the user name and password to be used.

The 'connect.def' system file located in the 'Setup' folder contains the connection definitions used by the PBVM (see page 30). Each connection definition has an alias that identifies the connection definition.

One of the connection definitions is specified as the default connection. The clone session automatically opens a connection to the server specified in the default connection. If a database is specified, then this is opened as well.

As a result of this, all subsequent sessions created by duplicating the clone also receive a connection to the server and the same open database.

The Item Stream Interface

Data is returned from the session runtime environment to the application server using the PrimeBase Item Stream Interface. The item stream is a first-in, first-out queue of typed values. Each item includes the data type, length, data and "flags". The flags indicate additional information about the item, such as whether the data is formatted, and whether the item is at the end of a row. PrimeBase Enterprise Objects uses special flags to indicate the type of the data, for example, if the item is part of a URL, and whether output conversion is necessary.

Items produced by code executed by the session are placed, in the order received, in the result page by the application server. More details of this process are provided in the chapter "Application Server Actions" on page 32.

Escape Commands

Escape commands are commands to the application server embedded in the item stream. The beginning of an escape command is indicated by a formatted NULL value of type VARBIN. This value can be placed in the output stream by using the PRINTF statement as follows:

```
printf((varbin) $null);
```

The PRINTF statement sets the format flag for the output items. An alternative is to use the PRINT statement as follows:

```
print/2 (varbin) $null;
```

The value after the slash character indicates the flags to be set. The flag 2 is the

format flag.

In general, escape commands have the following format:

```
<formatted-varbin-null> <name> <arguments>
```

The formatted VARBIN NULL value is followed by the name of the command, and then by a number of arguments, depending on the command.

Commands are instructions to the application server. For example a command is provided to send an HTTP re-direct to the browser, as follows:

```
print/2 (varbin) $null, "REDIRECT", "HTTP://www.prim-  
base.com";
```

PBAS supports numerous escape commands. A complete description of all escape commands is provided in Appendix "Escape Commands" on page 44.

Processing Page Requests

In general, a user agent, equipped with a URL, will request a source page from the application server. A URL or Universal Resource Locator refers to a particular page somewhere on the internet.

In addition to the page address, a user agent supplies information such as the request method (for HTTP: GET, POST, etc.), control information (HTTP header information such as cookie values) and data in the form of "search args" or "post data".

Search args are name/value pairs that are appended to the URL. Post data, is a block of data sent in the body of the request. HTTP header values specify the size and the content type of the post data.

In this section we discuss how the PrimeBase Application Server processes page requests.

The Operation Cycle

The processing of a page is divided into a number of stages and phases as follows:

- Pre-page stage: This is the first stage in the processing of a page. At this point it is known what the name of the page is that has been requested, and the type of the page request.
- Input phase: During this phase, input from the search args and posted data is input to the session.
- Pre-generation stage: This stage occurs after input, before generation of the result page begins.
- Header stage: This is the first stage of page execution. In this phase an HTTP header is written to the result page.
- Execution phase: During the execution phase, embedded code and LiveTags are executed to generate the contents of the result page. Static components in the page are copied to the result.

After completion of the execution phase, the result page is returned to the browser. "Actions" are used to specify exactly how a page will be processed by the Web application. How to specify actions is described in the chapter "Application Server Actions" on page 32.

Browser and Pool Sessions

The application server separates sessions into two categories. Browser sessions and pool sessions. Browser sessions are associated with a particular user agent. Pool sessions are allocated to a user agent for the duration of a page request only. After the page request has been processed, a pool session is returned to the pool

of sessions.

Whether a page is processed in the browser session or in a pool session depends on the page type. This information is specified in the page action definition as described in the chapter "Application Server Actions" on page 32.

If a page is to be processed by the browser session, then the application server uses the browser ID to determine the associated browser session. If no browser session exists then a new session will be created for the user agent. This means that the browser session is responsible for running the Web application instance for a particular user agent.

Pool sessions also run instances of the Web application. These instances, however, are not associated with a particular user agent. This means that pool sessions run "shared" application instances. Programmers must therefore be aware of the fact that session data is shared between user agents when using pool sessions.

Page Compilation and Caching

When a source page request is received the application server first determines the action to be used based on the page type. The type of the page is determined by the page extension (e.g. '.htm', '.htd', '.gif', etc.).

Pages that are processed using the EXECUTE action have an associated template document located in the application server 'Docs' folder.

If a page is to be executed the application server first checks to see whether the associated template has already been loaded into the page cache.

If not, the template document is read in from disk. It is then "compiled" by the application server. During compilation, the application server determines which parts of the page are static and which parts are dynamic.

Dynamic components are either in the form of embedded code in <PBT>, <DAL> or <SCRIPT> tags, or in the form of LiveTags. The dynamic components are passed on to the PrimeBase Virtual Machine for compilation and optimization. The template document, consisting of static components and compiled code is then placed in the application server page cache. Older pages will be removed from the cache if the specified size limit for the cache is exceeded.

Page Execution

After determining which session will be used to process the page, the application server allocates a thread to "execute" the page. During execution of the page the application server builds a result page.

Static components of the template document are copied directly to the result page. Compiled code, the result of dynamic components in the template document, is executed by the PrimeBase Virtual Machine. Values output to the item stream during execution are copied to the result page by the application server.

On completion, the result page is returned to the browser or user agent that made the request.

This section describes how to install the PrimeBase Application Server. Use the description below as a check list, and you will soon have the application server up and running.

3. Installing & Running

NOTE: *The PBAS can run without a Web server, and without a connection to a database or database system. If you are installing a second copy of the application server, or you wish to run more than one instance of it, please read "Running Multiple Instances of the Application Server" on page 15.*

Installing the Application Server

The software, whether on CD or downloaded from the internet, is "pre-installed". This means it is ready for use within minutes – you just have to copy the software to your machine, unpack and start PBAS.

Install the application server depending on your platform as follows:



Macintosh

Unpack the compressed file (usually 'PrimeBase Application Serv.sit' or 'PB_Application_Server.sit') and copy or move the folder containing PBAS (usually 'PrimeBase Application Server' or 'PrimeBase Application Serv') to the desired location on your harddrive.

If you have the regular PrimeBase CD distribution, it might contain a folder called 'PrimeBase Application Server' which you can use as an alternative instead of unpacking the archived version as outlined in the previous step.



Windows

Unpack the compressed file (usually 'PBAS.zip') and copy or move the folder containing PBAS (usually 'PBAS') to the desired location on your harddrive.

If you have the regular PrimeBase CD distribution, it might contain a folder called 'PBAS' which you can use as an alternative instead of unpacking the archived version as outlined in the previous step. However, if you copy the 'PBAS' directory from a CD, execute the program 'setwrite.exe' in this directory to reset the read-only attributes on all files before starting the application server.

On Windows NT/2000 you may install the application server as a service. To do this, start a Command Prompt ('cmd.exe') and change to the directory of the application server. Then enter the following:

```
assv.exe -install
```

Now you can start and stop PrimeBase Application Server via the 'Services' applet in Windows' control panel (use 'NET START <servicename>' and 'NET START <servicename>' in scripts and on the command line). To remove the service use the "-remove" option, when the PBAS service is not running.

NOTE: If you upgrade the PrimeBase Application Server on Windows NT/2000 you should first remove the older version of the service before you install the new version.



UNIX/Linux

Create a new user called "primebase" (or any name you prefer), and ensure that there is about 20 MB free disk space in the user's home directory. (Of course you can install PrimeBase using an existing user account as well.) Copy the file "pbas.tar.Z" to the user's home directory, uncompress and un-tar the contents:

```
uncompress pbas.tar.Z
tar -xf pbas.tar
```



Warning

For security reasons, do not place the PrimeBase Application Server and its subdirectories where it is accessible by the Web server (for example in the Web server document root directory, or the "cgi-bin" directory). This would make control files of the application server and pages served by it directly accessible from the Web.

Starting the Application Server

Macintosh

Start the PrimeBase Application Server by double-clicking on the application "pbas.acgi".



Windows

If you have installed the application server as a service on Windows NT/2000, then you may start the application server service via the 'Services' applet in the Windows control panel.



Otherwise, double-click on 'pbas.exe' in the PBAS directory and a console will appear. Once the application server has started you may quit the console by pressing CTRL-R, which leaves the application server running in the background. Using the Command Prompt or the target line of a shortcut you can start 'pbas.exe' with the '-b' option to start the application server in the background without a console:

```
pbas.exe -b
```

Whether the application server is running as a service or not you can bring up the PBAS console by double-clicking on 'pbcon.exe' in the PBAS directory.

You can shutdown PBAS by entering...

```
halt
```

...on a line by itself in the application server console, or by running the program 'pbas_stop.exe' in the PBAS directory.

UNIX/Linux



Run the 'pbas' executable. Similar to Windows, the '-b' option puts the application server in the background immediately, otherwise you can use CTRL-R to quit the console.

The UNIX/Linux console functionality is identical to that of the Windows console.

If you have downloaded the PrimeBase Application Server or installed it from a demo CD it runs for a limited period only. The activation information, including the expiry date, is stored in the file "pbas.key" in the PBAS directory.

Activating the Application Server

When the demo period has expired the application server will still start but return an error to any browser that accesses the application server.

As a registered user of PBAS you will be issued an "Activation Key" and a serial number, both of which are contained in a single file called 'pbas.key'. The activation key varies depending on the values assigned to the other activation parameters.

To activate PBAS, replace the 'pbas.key' file with the 'pbas.key' file containing your activation information. When the PBAS starts it reads the data from the 'pbas.key' file and stores the information in the corresponding activation system parameters. If the activation information is valid, it deletes the activation key from the 'pbas.key' file.

The following activation parameters will be set when you receive a new activation key:

- Activation Key – the activation key value issued to you.
- Serial Number – the serial number issued to you on registration.
- Registration Name – the name of the person or company to whom the applica-

tion server has been registered. For a demo version of PBAS the parameter is set to "Demonstration Server".

- Expiry Date – the current expiry date for PBAS. If you have purchased a full version of the application server this parameter must be set to "NeverExpire". Otherwise this is a string of the form: MM/01/YYYY. The application server will expire on the first day of the month specified.

NOTE: When you purchase the PrimeBase Application Server you must provide the name you would like to use as registration name.



Tip

If you are using the PrimeBase Application Server for development purposes only, you may request a free developer's key. To do this, send an e-mail to support@primebase.com. Please state the nature and duration of your project. We will send you a new 'pbas.key' file. To activate the application server, replace the file in the PBAS directory and restart the application server.

Accessing the Application Server from a Browser

To access the application server, start a browser, and enter the following URL:

```
http://<host-addr>:47120
```

<host-addr> is the IP address of the application server machine. You may use 'localhost' or '127.0.0.1' if the application server is running on the same machine as the browser. '47120' is the default port number used by PBAS. Upon this request, the application server will return the page 'appindex.lml', which is the default page.

This page presents a menu listing the installed application modules. The modules installed regularly include System and PBE Demo. Login is required to access certain modules. The default user name is "admin" with password "admin".



Warning

It is strongly recommended to change the admin password immediately to prevent unauthorized access to PBAS. You can set the admin password as follows: 1. click "System" to enter the "System Web Interface", 2. login as "admin", 3. click on "Owner", 4. click on "Administrator" and set a new password (for a detailed description see "The Owner Tab" on page 19).

Accessing the Application Server via a Web Server

To make the PrimeBase Application Server accessible via a Web Server the first thing to do is to install a Web server on the machine running the application server and check that you can access your Web server from a browser.

Note: PBAS has a built-in Web server, so installing a Web server is not really required for development. The internal Web server should be used for development purposes only in general, as it lacks most features common to more powerful Web servers.

The application server works with Web servers on the Mac that are compatible with the WebSTAR™ CGI or plug-in interface, and under Windows NT with servers

that have a Microsoft IIS compatible CGI or shared library interface. Under UNIX/Linux, the application server runs with Apache and Netscape Web servers amongst others, both as CGI or as Apache module.

To access the application server via a Web server, do the following:

Macintosh



Create an alias of 'pbas.acgi', and place it in your 'WebSTAR' folder, in a sub-folder called 'cgi-bin' – the 'cgi-bin' folder is not necessary, but is a convention on the Web that indicates use of a CGI application.

This description assumes you name the alias 'pbas.acgi', but you may name the alias anything you like as long as you use the '.acgi' extension, which is required by WebSTAR to indicate that the application server functions as an asynchronous CGI application.

URLs that refer to PBAS on the Mac will now take the form:

```
http://www.my_host.com/cgi-bin/pbas.acgi$/my_page.lml
```

NOTE: The "\$" sign is required by WebSTAR. If not provided in the URL, WebSTAR will not recognize the presence of a CGI.

Windows



Copy the executable 'primebase.exe' from the cgi-bin subdirectory of your PBAS directory to the directory designated as the CGI directory for your Web server – most often this directory is also called 'cgi-bin'.

You may rename 'primebase.exe' to whatever you like, but '.exe' is required by some Web servers, notably by MS IIS.

URLs that refer to the application server under Windows will take the form:

```
http://www.my_host.com/cgi-bin/primebase.exe/my_page.lml
```

Also 'primebase.dll', a dynamic link library version of the CGI which uses the MS IIS direct call interface, may be used in place of the 'primebase.exe'.

The DLL is installed and used in the same way as the '.exe' version. This means that the DLL must be placed in a virtual folder of IIS (or compatible Web server) which allows the execution of CGI programs. The virtual folder (which is a reference to an actual folder), and the executable permission must be setup using the IIS administration tool.

The name of the DLL or the '.exe' version of the CGI – by default "primebase" – must be the same as the name given to the application server. This is important especially if you wish to run multiple copies of the application server on one machine (see "Running Multiple Instances of the Application Server" on page 15).

The use of the 'primebase.dll' is not only more efficient than the 'primebase.exe' but also circumvents a bug in IIS (and Windows) that can cause your machine to run out of memory over time when using the 'primebase.exe'. The 'primebase.exe' CGI is still included for use with other Web servers that do not support the DLL interface used by IIS.

UNIX/Linux



Copy the executable 'primebase' from the cgi-bin subdirectory of your PBAS directory to the directory designated as the CGI directory for your Web server – most often this directory is called 'cgi-bin'. Set up file permissions correctly, so that the user context the Web server is run in has access to the 'primebase' CGI file. Often Apache for example is configured to run as user 'nobody'.

URLs that refer to the application server under UNIX/Linux will take the form:

```
http://www.my_host.com/cgi-bin/primebase/my_page.lml
```

To setup the PBAS Apache Server Module you will need to know the location of the Apache Server's configuration files and the location of the modules directory. You can find the modules directory location by looking in the configuration file 'httpd.conf' for a line beginning with 'LoadModule'. The line will look something like this:

```
LoadModule cgi_module /usr/lib/apache/mod_cgi.so
```

'/usr/lib/apache' will be the modules directory. Usually the configuration files can be found in '/etc/httpd/conf'. You will also need to be able to restart the Apache Web server. This can normally be done by entering the following command while being logged in as root/superuser on a SuSE Linux installation:

```
/etc/rc.d/apache restart
```

On a Red Hat Linux as root/superuser use this command to restart the Apache Web server:

```
/etc/rc.d/init.d/httpd restart
```

NOTE: If you cannot find these files or directories check with your system administrator on their locations and names.

Copy the file 'mod_pbas.so' from the cgi-bin subdirectory of your PBAS directory to the Apache server's modules directory.

Now you have to edit the Apache server's configuration file 'httpd.conf'. Assuming that your modules directory is '/usr/lib/apache/', search in the file for 'LoadModule'.

NOTE: If there are no lines beginning with 'LoadModule' then there is a good chance that your Apache server has not been configured to use dynamic shared objects (DSO).

You should see several lines beginning with 'LoadModule', the order of these lines is important. Look for this one...

```
LoadModule cgi_module /usr/lib/apache/mod_cgi.so
```

...and below it add the following line:

```
LoadModule PrimeBase_module /usr/lib/apache/mod_pbas.so
```

After adding the LoadModule line go to the location in the 'httpd.conf' file containing lines that start with 'AddModule' and insert this line:

```
AddModule mod_pbas.c
```

Apache now knows to load the PrimeBase module and must now be told when to call it. Assuming that the module is to be called with any URL starting with 'pbas' do the following:

Search the 'httpd.conf' file for the string 'Location'. You should see something like:

```
<Location /server-status>
  SetHandler server-status
</Location>
```

Here, add these lines:

```
<Location /pbas>
  SetHandler primebase-handler
</Location>
```

NOTE: 'primebase-handler' and 'PrimeBase_module' are the internal names of the handler and must be entered exactly as they are here, case being important.

Save the file and exit the editor.

Restart Apache by entering this command (or similar, see above) as root/superuser:

```
/etc/rc.d/init.d/httpd restart
```

NOTE: If Apache fails to start with an error that says something like "Invalid command 'LoadModule'" then your apache server has not been configured to use dynamic modules.

Now test the module by starting your browser and entering the following URL, where <host_address> is the IP address or host name of the Web server machine:

```
http://<host_address>/pbas/primebase/system/
```

You should get the same results as when you do the same via the CGI by entering the URL:

```
http://<host_address>/cgi-bin/primebase/system/
```

When using the module the component of the path that comes after 'pbas' must be the published name of the application server. To connect to an instance of PBAS published e. g. as PrimeBase2 you would enter the URL:

```
http://<host_address>/pbas/primebase2/system/
```

Enter the following URL in your browser, depending on what type of machine the Web server is running on:

Macintosh

```
http://<host_address>/cgi-bin/pbas.acgi$/index.htd
```

Windows

```
http://<host_address>/cgi-bin/primebase.exe/index.htd
```

UNIX/Linux

```
http://<host_address>/cgi-bin/primebase/index.htd
```

Accessing the Application Server via a Web Server from a Browser



<host_address> is the IP address or host name of Web server machine. When using the host name, some sort of host name resolution facility must be available at your site, for example DNS (Domain Name System).

'cgi-bin' is the name of the CGI directory (this may differ depending on the Web server configuration).

Check the PBAS console for error messages if you do not receive an error message in the browser.

NOTE: The Macintosh reference to the PBAS requires '.acgi\$'. The Windows reference requires '.exe', depending on the Web server, in particular, Netscape server will accept a CGI called 'primebase.cgi'.



Tip

If you use WebSTAR actions then the ".acgi\$" extension is not required. As a result, using actions gives you more flexibility in determining a URL that will work on all platforms.

4. Configuring the Application Server

You may control the numerous aspects of the PrimeBase Application Server using various system parameters and control/mapping files. In this section we discuss how to set some basic system parameters. For more details see the sections "Parameters" on page 22 and Appendix "System Parameters" on page 48.

Setting System Parameters

System parameters are stored in the PBAS environment file, called 'pbas.env', which is located in the 'Setup' folder (see page 28). You can use the "System Web Interface" (see "System Web Interface" on page 16) or the program "Environment-Edit" on the Mac, 'pbee.exe' under Windows or 'pbee' under UNIX/Linux to alter system parameters.

The application server can detect changes made to the 'pbas.env' file, and in this way these changes take immediate effect.

To view and edit system parameters of PBAS:



Macintosh

Double-click on the 'pbas.env' file. In the turning up parameters list you can double-click a line and change the corresponding value.



Windows

If it is not already there, place the program 'pbee.exe' in the same directory as the 'pbas.exe' program, and double-click the 'pbee.exe' to start the environment editor. Accessible environment files will be listed:

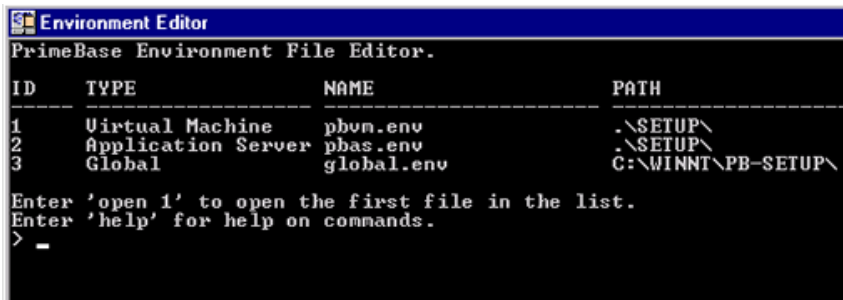


Figure 1: The PrimeBase Environment Editor (PBEE)

You need to open 'pbas.env' using the OPEN command followed by the number of the file. The LIST command gives you a complete list of editable parameters:

```
open 2
list
```

To get help on individual environment parameters enter the parameter id number and press ENTER:

```
152
```

:Use the HELP command to get a list of available commands

```
help
```

To set environment parameters to a new value, use the SET command followed by the parameter's ID and the desired value:

```
set [id] [value]
```

UNIX/Linux

If it is not already there, place the 'pbee' executable in the same directory as the 'pbas' executable, and start the 'pbee' program. The rest works the same as on a Windows system.

If you wish to run multiple copies of the application server on one machine, then each copy of PBAS must be given a unique name (do this by setting the system parameter 302 "Server Name", see page 48).

In addition to a unique name, each copy of the application server must have a unique URL. If you are accessing the application server directly via its built-in Web server, then each server will use a different port number. Since the name of the application server by default is used to determine the port number used, this is generally the case. Alternatively, you can set the port number of the application server with the system parameters 2500 "HTTP Port Number" and 2501 "HTTP Usage" (see page 52).

If you are accessing the application server via a Web server, then do the following:

Macintosh

Create an alias, each with a different name, in the Web server folder (or in the 'cgi-bin' folder if you are using one) for each copy of the application server.

Windows

Place one copy of 'primebase.exe' in the Web server's "cgi-bin" directory for each copy of the application server. Set the name of the executable to the name of the



Running Multiple Instances of the Application Server



corresponding application server (followed by '.exe').



UNIX/Linux

Place one copy of the 'primebase' executable in the PBAS 'cgi-bin' folder in the Web server's 'cgi-bin' folder for each installed instance of PBAS. Set the name of the executable to the name of the corresponding name of the application server.

Control/Mapping Files

Control/Mapping files are text files located in the 'Setup' folder of the PrimeBase Application Server (see "'Setup' Folder" on page 28) which control various aspects of its behavior. The text file format is not platform specific.

NOTE: All control/mapping files are read once by the application server on start-up. If a mapping file is changed, you must restart the application server for the change to take effect.

The following is a list of PBAS control/mapping files:

- **Character Conversion**

The character conversion mapping files, 'convchar.in' and 'convchar.out', determine how the application server converts characters received from the Web, and how it converts characters placed in HTML pages sent to the Web. How the application server converts characters is explained in detail later in the section "Character Conversion" on page 31.

- **Action Map**

The Action Map control file, called 'action.map', tells the application server what types of files it may return to the Web user and what it must do before returning the page. The Action Map is described in the chapter "Application Server Actions" starting on page 32.

- **LiveTag Maps**

A LiveTag Map file maps HTML tags to PBT procedure or method calls. The name of the LiveTag Map file is specified in the Action Map. The application server provides a great degree of flexibility in specifying the mapping of HTML tags to PBT procedure calls. LiveTags and the LiveTag Map file is explained later in the section "LiveTags" starting on page 39.



Warning

When you start PBAS after changing a mapping file, be sure to check the console or log file 'pbas.log' for any errors that may have occurred while reading the file. If an error occurred, the mapping you have specified will not work correctly.

5. System Web Interface

To access the PrimeBase Application Server, start a browser, and enter the following URL:

```
http://<host_address>:47120
```

<host_address> is the IP address of the application server machine. You may use 'localhost' or '127.0.0.1' if the PBAS is running on the same machine as the browser. '47120' is the default port number used by PBAS. Upon this request, the application server will return the page 'appindex.html', which is the default index page.

This page presents a menu listing all installed application modules of PBAS even if

their access is restricted.

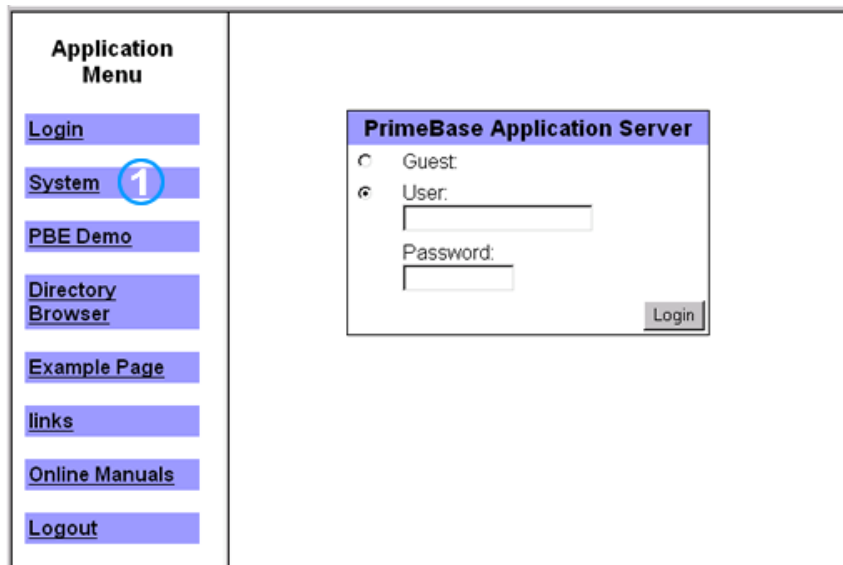


Figure 2: The default index page of the PrimeBase Application Server

To change the index page you can replace the 'index.htd' or the 'appindex.htd' file in the PBAS 'docs' directory or change the redirect command call in the 'index.htd' file. If you want to make a module's index page the PBAS default page, place a '*' in front of the name of the module's default page. The module default page is specified in the module's PBT source file, in the line which declares the module itself (for example: `PBE:declareModule("DemoModule","PBE Demo","*index.htd", "pbedemo", "pbedemo!Common.Employees", "login,password");`).

To open the "System Web Interface" use the 'System' link (1) in Fig. 2. The system module's index page provides five tab panels (see Fig. 3):

System Module Overview

- **"Application":**
An overview of the installed modules as they are on the default PBAS index page, but – in contrast to the default PBAS index page – only accessible modules are displayed.
- **"Layout":**
To browse, organize, upload and delete HTML files and pictures belonging to the installed application modules.
- **"Programmer":**
To browse, organize, upload, delete and edit program files of the installed application modules.
- **"Owner":**
To set up users and to define privileges for the various modules of the application server.
- **"Configuration":** To modify configuration settings and system parameters.

Since login is required for certain modules in the different sections only those modules can be seen for which access is granted. To login as administrator with unlimited access rights use the default user name "admin" with password "admin". To open the system module of PBAS enter the user name (1) and password (2), and press the login button (3) in Fig. 3.

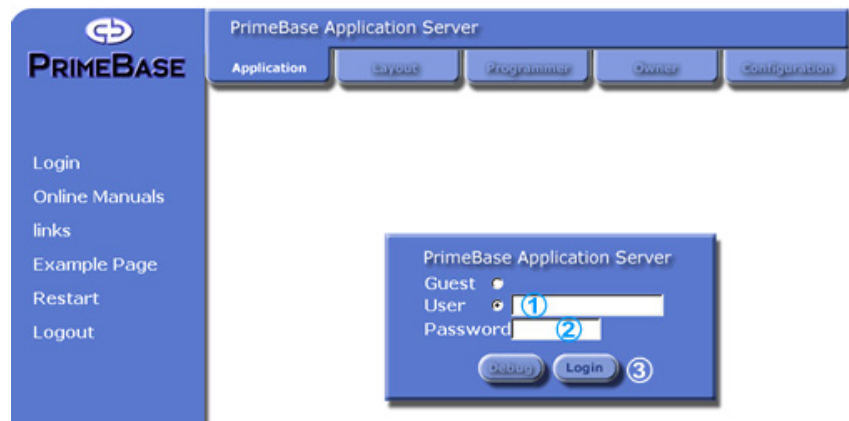


Figure 3: The PBAS "System Module"

The Application Tab

The fields in the "Application" tab allow you to access the application modules. This section is intended to make it possible for you to work with the modules or test them in development phase.

Login

„Login" can be used to login PBAS as a user. This login always refers to a system logon, so that all possible access rights to every installed module is provided.

Online Manuals

At "Online Manuals" you can access the different PrimeBase online manuals and PDF documentations shipped with the PBAS. Here you can learn more about PrimeBaseTalk (PBT) and the PrimeBase Enterprise Objects (PBE0).

Example Page

The example page is a simple demonstration of LiveTags to show developers how they work in the context of PBE0.

Directory Browser

The directory browser is an example module for PBE0 that also allows you to browse the files of the PBAS via a Web browser.

PBE Demo

The "PBE Demo" is a complete application module with database access. To use it the first time you have to connect the PBAS with the PrimeBase SQL Database Server (see "Connections" on page 20). This demo application is designed to give you a standard application with database connectivity, tables, relationships between the tables and a Web front-end.

Restart

Restart the PBAS. Programmers often have to restart the PBAS for changes made to code files of the different modules to take effect. After a restart all system and configuration files are loaded again as well as the code files of the various modules.

Logout

After working with the system module – especially as an administrator with full access – you should logout here.

In the "Layout" section choose one of the listed application modules on the left side frame to get a complete list of associated files in the main frame. Of course only those modules are shown in the list the logged-on user has the specific rights for. It is possible to up- and download files and to rename or delete them.

The Layout Tab

The "Programmer" section works similar to the "Layout" section with the difference that you need programming privileges and will get access to the code files of the installed modules instead of the design related files. Besides the functions supported in the layout section in this section you can also edit files directly and initiate compilation of all application modules.

The Programmer Tab

PrimeBase Application Server

Application Layout Programmer Owner Configuration

Users for Online Manuals:

Name	Login	E-mail	SA	Privileges
Chief	he		\$FALSE	Owner
Guest	we		\$FALSE	User
Designer	me		\$FALSE	Layout
Programmer	you		\$FALSE	-
Administrator	admin		\$TRUE	-
New				

ID: 1

Name: Administrator E-mail:

Login: admin Password:

SA: Privileges:

Cancel Save

Figure 4: The "Owner" section of the PBAS "System Web Interface"

In the "Owner" section you get a list of all registered users, where you can select an existing user (1 in Fig. 4) or add a new user. To add a new user you have to press "New" (2), enter a name, a user login and password and save the dataset (3). Check "SA" if you want to provide full access to all modules. In addition you can choose an application module in the left frame (4) and assign or change its privileges (5).

The Owner Tab

Tip

The first modification you should perform because of security reasons is a change of the default administration password.



Different privileges can be set for the installed application modules. Possible modes are "User", "Layout", "Programmer", and "Owner", which are hierarchic, latter modules inherit all the rights of the previous module. "User" gives you no rights to change any files, "Layout" allows you to change the HTML files and pictures, "Programmer" enables you to change code files and "Owner" to setup new users and assign access rights for the module.

All settings and changes are stored in the 'users.txt' file in the PBAS 'Setup' folder (see page 28).

The Configuration Tab

In the "Configuration" section you can modify the deployment mode, the connection setting, and the system parameters.

Deployment

The deployment mode can be set to production, development or debug.

Production – Set the PBAS operation to production when the Web site is no longer under development. In production mode the execution of your application is optimized.

Development – This mode is for normal multi-user development operation of the PBAS.

Debug – Debug mode can be used during development of a Web site when access to the Web site is restricted to fully authorized personal. In this mode a "debug" login is allowed which gives users full access to all aspects of the PBAS without requiring a user name or password.

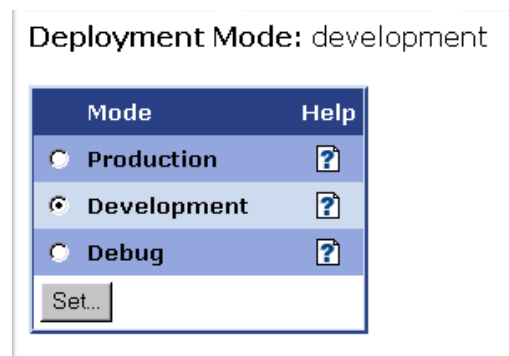


Figure 5: Set the deployment mode

Since deployment mode is stored as system parameter 2600, it can also be changed in the parameters menu of the "Configuration" section (see "Parameters" on page 22) or – especially if you do not have access to any browser – directly with the PrimeBase Environment Editor (PBEE; see sub-section "Main Directory" on page 23).

NOTE: In development mode the values for "Page Cache Size" and "Open Sessions Ready" are ignored to ensure that the changes made during development will take effect immediately.

Connections

With this option you can configure the connections used by PBAS to connect to a database server. The application server requires a connection definition, which is stored in a text file called 'connect.def' in the PBAS 'Setup' folder (see page 28) to do this. To create a connection the default values in the top line of 'connect.def' are used. In the system module you can modify an existing connection definition, create a new one and redefine the default connection.

Three different connection definitions are shipped with the PBAS (Fig. 5):

- *no_server* – The application server has no connection to any server or database but to the PrimeBase Virtual Machine. This is the pre-installed default connection of PBAS.
- *local_server* – The application server supports shared memory protocol and is connected to the PBVM and the PrimeBase SQL Database Server (PBAS).
- *remote_server* – The application server supports TCP/IP protocol on host address '127.0.0.1' and has a connection to PBVM and PBDS.

Connection List:

Default	Alias	Extension	Protocol	Host/Zone	Server	Database
<input type="radio"/>	no_server (2)	PBVM.DLL	-			(3)
<input checked="" type="radio"/>	local_server	PBVM.DLL	Shared Memory		PrimeServer	
<input type="radio"/>	(1) remote_server	PBVM.DLL	TCP/IP	127.0.0.1	PrimeServer	

New

Figure 6: The „Connection Setup“ of PBAS

In the definition setting menu you can set the default connection (1 in Fig. 6), edit an existing connection by pressing the link (2) or the pen icon or delete it by pressing the trash can icon (3).

Alias: (1)

Extension: Virtual Machine (2) Database Runtime PBVM.DLL

Protocol: (3)

Type: Database Server (4) Open Server Server: PrimeServer (5)

Database: (6) User: (7) Password: (8)

Cancel Save

Figure 7: Change the "Connection setup"

In the connection definition you specify the following information:

- A name to identify the connection (see 1 in Fig. 7).
- The type of virtual machine used to create a session, either the standard virtual machine, or the server runtime virtual machine (2).
- The protocol to use to connect to the server (3) and protocol specific information (an additional input field will open e.g. for TCP/IP, host address).
- The type and the name of the database server to connect to, in other words the PBDS or the PrimeBase Open Server (PBOS), if you want to connect to 3rd party database (4).
- The name of the database server (5).
- The name of the database to be opened (6).
- A user name and password for the database server (7).

The type of virtual machine must be specified, but further information is optional. For example, if no server is specified, the application server will work but not connect to a server. If no database is specified, the PBAS will not open any database.

Warning

If you have specified a server, do not forget to specify a user name and password. A connection will not be established unless a user is specified.























NOTE: When setting up a connection, check to see whether the list of databases is available on the server. If not, the server is not running, or the connection information you have provided is incorrect.

Since the default connection alias is stored in system parameter 2706, it can also be changed in the parameters menu of the „Configuration" section (see below) or if you do not have access to a browser – directly with the PBEE.

Parameters

In the "Parameters" menu you can make changes on system parameters to affect the behavior of PBAS. The possible settings range from general configuration and activation parameters to session time-outs and error page settings.

Parameter List:

ID	Name	Modified	
300	HTML Page Location	docs	 
302	Server Name	PrimeBase	 
303	Serial Number	DVAAB400301402	 
304	Registration Name	Demonstration Server	 
307	Registered Maximum Connections	0	 
308	Maximum Open Sessions	20	 
314	Activation Key	B9MQ-GX0Z-Z4LT	 
315	Expiry Date	4/2001	 
317	317	7200	 
330	Engine Total	1	 

ID: 302

Name: Server Name

Value:

The name of the server.

Figure 8: The "Parameter List" of PBAS

You have to press one of the ID links in the "Parameter List" (1) or a pen icon (2) to open a panel where you can change the value of the corresponding setting (3) and save it (4). On pressing the question mark icon (2) you will get a brief description which explains the parameter.

Different types of setting can be modified (for details see appendix "System Parameters" on page 48):

General Configuration – Main parameters such as HTML page location or application server name can be set.

Session Creation, Configuration and Time-outs – Sessions are an important aspect of the PrimeBase Virtual Machine (see "PrimeBase System Overview"), which are initiated by PBAS. The handling of PBVM sessions by PBAS can be set, especially the quantity and duration of sessions.

Protocol Logging/Display – PBAS creates protocol files for logging errors, warnings, etc. File name, protocol and display level can be set.

Identification of Browsers – Incoming request can be handled in different ways. To ensure a connection between the user's browser and PBAS, it is possible to choose different methods depending on whether you wish to use cookies or not and which type of cookie should be used.

BLOB Files and URL parameters – With PBAS Binary large objects (BLOBs) can

be generated temporarily and printed in HTML pages. For temporary files a directory path and file name has to be specified as well as maximum disk space and cache time-out.

Activation Parameters – License data of the PBAS are stored as system parameters. All data of a valid activation key can be entered via the system module.

HTTP Controls – Also HTTP specific parameters such as port number can be set for PBAS.

Error Page Settings – You can choose another error page for PBAS. This capability allows you to format the error messages returned to the browser.

Reconnect

In the "Configuration" section you can reconnect the PBAS according to the connection definition set in the "Connection" menu.

Tip

In production mode of PBAS you may be interested to delete the system web interface to prevent unauthorized access on system settings. Since the system interface itself is a application module you just have to delete or displace the 'system' subdirectory in the PBAS 'docs' folder.



In this chapter the PrimeBase directories and their contents are described.

6. Directories and Files

In the main directory of PBAS the executable files are stored as well as release notes and a readme text. The three subfolders of the main directory are described in the following sections.

Main Directory

PrimeBase Application Server Executable

Macintosh:	<code>primebase.acgi</code>
Windows:	<code>primebase.exe</code>
Unix/Linux:	<code>pbas</code>

The PrimeBase Application Server executable.

Start Server Shell Scripts

Macintosh:	<code>n. a.</code>
Windows:	<code>n. a.</code>
Unix/Linux:	<code>start_standard_server</code>

Scripts to start the enterprise or standard application server in the background.

Stop Server Shell Script

Macintosh:	<code>n. a.</code>
Windows:	<code>n. a.</code>
Unix/Linux:	<code>stop_application_server</code>

A script to stop the application server.

Server Console Shell Script

Macintosh:	n. a.
Windows:	n. a.
Unix/Linux:	application_server_console

A script to connect an interactive console to the application server running in this location.

Service Executable

Macintosh:	n. a.
Windows:	assv.exe
Unix/Linux:	n. a.

An executable used to install the application server as a service with Windows NT or 2000.

Stop Server Executable

Macintosh:	n. a.
Windows:	pbas_stop.exe
Unix/Linux:	pbas_stop

An executable used to shutdown the application server.

PrimeBase Console

Macintosh:	n. a.
Windows:	pbcon.exe
Unix/Linux:	pbcon

The PrimeBase Console is a user-interface to all PrimeBase server applications running on your machine.

PrimeBase Application Server Engine

Macintosh:	n. a.
Windows:	ase.exe
Unix/Linux:	ase

The PrimeBase Application Server Engine (ASE) is a version of the PBAS started by the PrimeBase Enterprise Application Server (PBEAS). The PBEAS will start and monitor multiple instances of the ASE.

PrimeBase Enterprise Application Server

Macintosh:	n. a.
Windows:	pbeas.exe
Unix/Linux:	pbeas

The PrimeBase Enterprise Application Server acts as a controller and starts or shuts down the application server engines (ASE) as required. The PBEAS also allows you to monitor the activity of the ASEs and start or stop them manually.

PrimeBase Virtual Machine

Macintosh:	PrimeBase VM Library
Windows:	pbvm.dll
Unix/Linux:	libpbvm.so

The PrimeBase Virtual Machine linked as a shared object, which is loaded by the application server when a session is created.

PrimeBase SQL Database Server

Macintosh:	PrimeBase DS Library
Windows:	pbds.dll
Unix/Linux:	libpbds.so

The PrimeBase SQL Database Server linked as a shared object. The PrimeBase DS Library is loaded when a runtime server connection is made.

Environment Editor

Macintosh:	Environment Edit
Windows:	pbee.exe
Unix/Linux:	pbee

The PrimeBase Environment Editor is used to set system parameters stored in the '*.env' files of the PBAS 'Setup' folder.

Automation Client

Macintosh:	Automation Client
Windows:	pbac.exe
Unix/Linux:	pbac

The PrimeBase Automation Client (PBAC) is used to automatically perform various tasks. PBAC has several modes including interactive, execute and scripting.

Connection Setup

Macintosh:	Connection Setup
Windows:	n. a.
Unix/Linux:	n. a.

This tool enables you to change the connection settings stored in the 'connect.def' file. Connection setup can also be changed with the automation client.

Shared Memory Manager

Macintosh:	n. a.
Windows:	n. a.
Unix/Linux:	smm

The shared memory manager executable (SMM) is started when the shared memory resources used by PrimeBase are first allocated and continues to run until there are no more processes attached to the shared memory. Its primary function is to maintain a real-time clock in shared memory used by all applications attached to the shared memory.

Registration Key

Macintosh:	pbas.key
Windows:	pbas.key
Unix/Linux:	pbas.key

The pbas.key file contains all PBAS activation information.

Read Me

Macintosh:	Read Me!
Windows:	readme.doc
Unix/Linux:	readme.pbas

In the readme file you can find a short introduction into the PrimeBase System, some installation notes and a short description of contents.

Release Notes

Macintosh:	release.txt
Windows:	release.txt
Unix/Linux:	release.txt

The release.txt contains release notes with explanations to all modifications and additions that have been made for the current release of PBAS.

Documentation

Macintosh:	documentation.html
Windows:	documentation.html
Unix/Linux:	documentation.html

The top level index page into the PrimeBase Application Server's documentation.

Setwrite Tool

Macintosh:	n. a.
Windows:	setwrite.exe
Unix/Linux:	n. a.

Run the setwrite.exe if you have copied PBAS from a CD using the Windows File Manager to set READ/WRITE access on all files.

WebSTAR Plug-in

Macintosh:	Plug-in
Windows:	n. a.
Unix/Linux:	n. a.

The PrimeBase WebSTAR server plug-in module can be used in place of the 'pbas.acgi' when using the WebSTAR Web Server. It then has to be copied in the Web server's plug-ins folder.

'Cgi-bin' Folder

Macintosh:	n. a.
Windows:	\cgi
Unix/Linux:	/cgi

The cgi-bin directory contains the Web server communication tools of PBAS. Its contents are described below in this chapter. For Macintosh the plug-in is stored in the main directory.

'Docs' Folder

Macintosh:	:docs
Windows:	\docs
Unix/Linux:	/docs

The 'Docs' folder is the PBAS document root directory. Its contents are described below in this chapter.

'Setup' Folder

Macintosh:	:setup
Windows:	\setup
Unix/Linux:	/setup

The 'Setup' folder contains environment and configuration files as well as the source code of the PrimeBase Enterprise Objects framework. Its contents are described below in this chapter.

The cgi-bin directory contains the Web server communication tools of PBAS.

'Cgi-bin' Folder

PrimeBase CGI Application

Macintosh:	n. a.
Windows:	primebase.exe
Unix/Linux:	primebase

The CGI application started for each request by the Web Server. It sends the request to the application server and passes the reply back to the Web Server.

Apache Server Plug-in

Macintosh:	n. a.
Windows:	n. a.
Unix/Linux:	mod_pbas.so

The Apache server plug-in module for PrimeBase can be used in place of the CGI when using the Apache Web Server version 1.3.12 or later. It is loaded once by Apache at start-up and then handles the sending of HTML requests to the application server and passing replies back up to the Apache server. It has the advantage over the CGI in that a process isn't started for each HTML request and that the connection to the application server is maintained.

ISAPI Interface

Macintosh:	n. a.
Windows:	primebase.dll
Unix/Linux:	n. a.

This dynamic link library version of the CGI works with the MS IIS direct call interface. It may be used in place of the primebase.exe. It has the advantage over the CGI in that a process isn't started for each HTML request and that the connection to the application server is maintained.

The 'Docs' folder is the PBAS document root directory. This directory contains the documents executed and served by the application server. All application modules of PBAS are stored in a sub-folder of this directory. All of these sub-folders have an 'exec' folder containing the source code of each module.

'Docs' Folder

Default Source Document

Macintosh:	index.htd
Windows:	index.htd
Unix/Linux:	index.htd

The default document served by PBAS. Other files in the 'Docs' folder, such as 'login.lml' or 'timeout.htd', are the default PBAS documents for different requests or tasks.

PBE Demo

Macintosh:	:demo
Windows:	\demo
Unix/Linux:	/demo

The 'demo' sub-folder contains the "PBE Demo" – a complete application module with database access.

Directory Browser

```
Macintosh:      :dir
Windows:        \dir
Unix/Linux:    /dir
```

The directory browser in the 'dir' sub-folder is an example module for PBEO that allows you to browse the host file system via a Web browser.

Example Page

```
Macintosh:      :eg
Windows:        \eg
Unix/Linux:    /eg
```

The example page in the 'eg' sub-folder is a simple demonstration of LiveTags to show developers how they work in the context of PBEO.

PrimeBase Manuals

```
Macintosh:      :manual
Windows:        \manual
Unix/Linux:    /manual
```

In the 'manual' sub-folder you can access the different PrimeBase online manuals and PDF documentations shipped with PBAS. Here you can learn more about PBT and PBEO.

System Module

```
Macintosh:      :system
Windows:        \system
Unix/Linux:    /system
```

The 'system' sub-folder contains the "System Web Interface" which has been described in detail in the previous chapter.

PrimeBase Tutorial Demo

```
Macintosh:      :tutdemo
Windows:        \tutdemo
Unix/Linux:    /tutdemo
```

In the 'tutdemo' sub-folder you can find the templates for the "PrimeBase Tutorial".

'Setup' Folder

The PBAS 'Setup' folder contains environment and configuration files as well as the source code of the PrimeBase Enterprise Objects framework.

Connection Definition

```
Macintosh:      connect.def
Windows:        connect.def
Unix/Linux:    connect.def
```

The connection definition file defines connections to various servers used by PBAS. The top-most definition is used by default. See also the section "Connection Definition" on page 30.

Action Map

```
Macintosh:      action.map
Windows:        action.map
Unix/Linux:    action.map
```

The action map is a text file which specifies the actions for each type of page. Actions entered in this file apply to all pages returned by the application server.

See also the chapter "Application Server Actions" on page 32.

Livetag Map

Macintosh:	<code>pbetag.map</code>
Windows:	<code>pbetag.map</code>
Unix/Linux:	<code>pbetag.map</code>

The livetag map is a text file which specifies which tags of an source document should be handled as Livetags. See also the chapter "LiveTags" on page 39.

PBAS System Parameters

Macintosh:	<code>pbas.env</code>
Windows:	<code>pbas.env</code>
Unix/Linux:	<code>pbas.env</code>

The PBAS environment file contains system parameters that can be changed by using the PrimeBase Environment Editor (see page 14) or the "System Web Interface" (see page 16).

PBVM Parameters

Macintosh:	<code>pbvm.env</code>
Windows:	<code>pbvm.env</code>
Unix/Linux:	<code>pbvm.env</code>

The PrimeBase Virtual Machine environment file contains settings and parameter values.

Input Character Conversion

Macintosh:	<code>convchar.in</code>
Windows:	<code>convchar.in</code>
Unix/Linux:	<code>convchar.in</code>

The input character conversion map is used to convert characters from the Web to the database character set. See also the subsection "Character Conversion" on page 31.

Output Character Conversion

Macintosh:	<code>convchar.out</code>
Windows:	<code>convchar.out</code>
Unix/Linux:	<code>convchar.out</code>

The output character conversion map used to convert characters stored in the database to Web format. See also the sub-section "Character Conversion" on page 31.

Registered Users

Macintosh:	<code>users.txt</code>
Windows:	<code>users.txt</code>
Unix/Linux:	<code>users.txt</code>

'Users.txt' is the file where the registered users and their access privileges are stored.

Protocol

Macintosh:	<code>pbas.log</code>
Windows:	<code>pbas.log</code>
Unix/Linux:	<code>pbas.log</code>

'pbas.log' is the default protocol log file of PBAS. This file will be generated automatically if it doesn't already exist.

'Exec' Folder

```
Macintosh:      :exec
Windows:        \exec
Unix/Linux:    /exec
```

The 'exec' folder is a sub-folder of the 'Setup' directory and contains system and startup code used to boot Web applications.

'Initialize' Folder

```
Macintosh:      :initialize
Windows:        \initialize
Unix/Linux:    /initialize
```

The 'initialize' folder is a sub-folder of the 'Setup' directory and contains code loaded when the virtual machine is initialized. Initialized code is shared by all sessions.

'Startup' Folder

```
Macintosh:      :startup
Windows:        \startup
Unix/Linux:    /startup
```

The 'startup' folder is a sub-folder of the 'Setup' directory and contains code loaded when a session is created.

'Unicode' Folder

```
Macintosh:      :unicode
Windows:        \unicode
Unix/Linux:    /unicode
```

The 'unicode' folder is a sub-folder of the 'Setup' directory and contains the files used for UniCode translation used by the PrimeBase Virtual Machine.

'Custom' Folder

```
Macintosh:      :custom
Windows:        \custom
Unix/Linux:    /custom
```

The 'custom' folder is a sub-folder of the 'Setup' directory and can be used for one's own code files that are to be initialized in addition to the default files.

Connection Definition

The connection definition file 'connect.def' specifies the connections to various servers used by PBAS. This file consists of a number of lines of text. Each line defines a connection. The topmost line is the default connection, which the application server uses to log on to the server.

The 'connect.def' can be changed by using the "System Web Interface" (see "Connections" on page 20) or via a text editor or with the PrimeBase Automation Client.

Each line of the 'connect.def' has the following format:

```
[alias]:[protocol]:\[option1]\[option2]{\...
```

<alias> is the name of the connection. This can be any name you like. For compatibility reasons, the alias should not contain spaces but underscores.

<protocol> is the name of the communications protocol to be used. On windows this can be:

na – Not announced.

tcp – TCP/IP protocol, used to communicate over the network.

ipc – Shared memory communications for a fast local connection to a server.

mem – shared memory.

<options> are the various options for this connection definition. All connection options have the form:

```
\x<value>
```

where "x" indicates the type of option, followed by the value.

This is a list of the available option types:

\ef – The virtual machine.

\n – The server name.

\su – The user name to be used to log-on to the server.

\sp – The password of the user.

\b – The name of the database to be opened by the application server.

\z – The server machine IP address or host name (if you are using a name server). This option is only required for TCP/IP connections.

Example

This is a connection definition, called "MyConn", used to connect to a server called "PrimeServer", on a machine with the IP address "120.191.1.34":

```
MyConn:tcp:\efPBVM.DLL\nPrimeServer\sadministrator\sprose-
bud\bPrimeBase\z120.191.1.34
```



One feature of the PrimeBase Application Server is its ability to handle foreign language characters as specified by the HTTP standard. The HTTP/HTML standard specifies that certain characters are encoded when sent over the Web. Encoding is different depending on whether data is being output to the Web, or input from a browser.

For this purpose PBAS uses two files which describe the conversion of characters to and from the Web. The files must be placed in the PBAS setup folder/directory.

The 'convchar.out' file describes the mapping of characters that are output to the Web. All character data output by PBAS is converted according to the mapping specified in this file. This file contains entries of the form:

```
'<' [CHAR] '>' [STRING]
```

where <[CHAR]> is a single character, or a decimal value representing the character. For example <A> is the same as <65>, because the ASCII value of "A" is 65. [STRING] is a sequence of characters that should replace the character on the left on output. This is normally a sequence like "ã" which is a sequence of characters defined by HTML which represent an "a" with a tilde on top. The quotes (single or double) are optional unless the sequence includes spaces or "<" or ">". For example you may map the trademark character to sequence "TM". Characters not appearing in the "convchar.out" file are not converted.

The 'convchar.in' file describes the mapping of characters on input. Encoded values received are converted according to the mapping in this file. This includes values entered into forms. These values are encoded by the browser before being sent to the Web server. This conversion takes in account the native character set on which the browser is running. Entries in this file are of the form:

```
[HEXCHAR] '<' [CHAR] '>'
```

Character Conversion

where [HEXCHAR] is the hex type encoding used by the browser: a "%" character followed by two hex digits. [CHAR] is single character or decimal value as described above. Encoded values that do not appear in the map are converted to characters with the given hex value. In general this is OK for any character with ASCII value 127 or less. These characters are the same no matter which character coding standard is used (ASCII, ANSI, etc.). For example %20 does not need to be explicitly mapped to a space because the value 32 is (hex 20) is a space no matter what character set is being used.

The PrimeBase Application Server is shipped with default character conversion files. The default mapping is for the ASCII character set used by the Macintosh. This is also the native character set used by the PrimeBase Database Server, and by the PBT interpreter.

You only need to change the default mapping if you are using the Application Server in conjunction with a 3rd party DBMS that is using a different character set. The character set used by PBAS should match the character set used by the DBMS, or foreign language characters will not be sorted and compared correctly by the DBMS. Change the "convchar.in" and "convchar.out" files as required. The application server will print an error on the console if it fails to parse a "convchar" file, but not if it does not find the file at all.

7. Application Server Actions

Actions determine how the PrimeBase Application Server processes pages requested by the browser or user agent. Different actions can be specified for different page types. The page type is determined by the extension.

A request page, also known as a source page, is usually associated with a document located in the application server's 'Docs' folder (see page 27). Therefore, if a source page has an associated document then the action determines the operation to be performed on the document.

For example, some documents are simply returned "as is" to the browser. This is appropriate for graphics (.gif, .jpeg, etc.) and documents containing various static data, such as PDF files. Other documents that generate dynamic content are compiled and executed by the application server. The resulting output is then returned to the browser as the result page.

NOTE: Documents will not be returned by the application server if an action has not been specified for the page type of the document. An attempt to retrieve such a document will result in a privilege violation error.

This chapter describes the types of actions supported by PBAS and various other options for the processing of web pages.

Action Maps

Action maps are text files which specify the actions for each type of page. Action map files have the name 'action.map'. There are two types of action maps: Global action maps and module specific action maps.

The global action map is located in the application server 'Setup' folder. Actions entered in this file apply to all pages returned by the application server. Module specific action maps are located in the modules 'exec' folder. Module specific action maps apply to all pages in the module. The actions in the module action map override the actions specified in the global action map.

Each line in the action map specifies the action to be applied to a specific type of page and is called an action declaration. The format of an action declaration is as follows:

```
<page-extension> <content-type> <action-specification>
```

- `<page-extension>` specifies the type of page to be processed. For example: `.html`, `.gif`, `.css`, etc.
- `<content-type>` specifies the HTTP content of the data returned, for example: `text/html`, `image/gif`, `text/css`. The content type is specified in the HTTP header of the reply to the browser.
- `<action-specification>` is defined as follows:
`<action-type> ['[' <action-options> ']'] { '/' <execution-option> }`
 In other words: an action specification is an action type followed by optional action options followed by any number of execution options. Action options must be placed in square brackets, and are separated by a semi-colon (;). Execution options are preceded by a slash (/).

Here is an example of an action specification:

```
.exec text/html EXECUTE [ INPUT=Web:input ] /BROWSER/HTML /NOBREAK
```

This line specifies the action for `.exec` type pages. The action type is `EXECUTE`. The `INPUT` action option has been set to the value `Web:input`. The execution options `BROWSER`, `HTML` and `NOBREAK` have also been specified.

The various action types, action options and execution options are described in detail in the sections below.

The page action type determines how the page is processed by the application server. The following types of actions may be specified for a page:

Types of Actions

EXECUTE

Pages processed using the `EXECUTE` action require an associated template document located in the application server's `Docs` folder.

As explained in the chapter "Key Concepts" (see page 1), the `EXECUTE` action processes a page by compiling and loading the template document into the application server page cache. The page is then executed to produce a result page that is returned to the browser.

Template documents contain static and dynamic components. The dynamic components are either in the form of embedded code or `LiveTags`.

In order to use `LiveTags` a `LiveTag` map can be specified for the page. This is done with the `LIVETAGS` action option described below.

RUN

Source pages consisting entirely of code can be run. These pages must have an associated document in the application server's `Docs` folder. On request, if the document is not already in the application server page cache, it is loaded into RAM and compiled by the PrimeBase Virtual Machine. The resulting code is stored in the application server page cache. The code is then executed by the PBVM. Output generated during execution forms the result page that is returned to the browser on completion.

RETURN

This action returns the requested document to the browser without any modifications. In other words, the `RETURN` action is used to return static data to the browser. Documents processed by the `RETURN` action are not placed in the application server page cache. Please note that, in production, the job of returning static data to the browser is best done by the Web server. Returning static data from the application server is an unnecessary load on the application server. This means that you should configure your Web site so that static data is returned by the Web server.

NOACCESS

This action returns an access violation to the browser if a source page with the specified extension is requested. This is similar to the default behavior for pages requested that have no entry in an action map, with one significant difference. Source pages processed by the NOACCESS action generally do not have an associated document in the application server's 'Docs' folder.

Using various action options, such as PREPAGE (see below), it is possible to "intercept" the request and return a reply to the browser directly out of the program, before an error is generated. For example, if you have static data files that you wish to return to the user, but you want to check whether the user may download the file, then the NOACCESS action can be used. In the PREPAGE callback you can check the user's privileges and return the file using the RETURN escape command. If the user does not have the privileges to access the file, then the program exits the PREPAGE callback normally, and an access violation error is returned to the user.

Action Options

Action options specify the details of how a page should be processed. All action options, except LIVETAGS, specify "callback functions" which are executed at various stages/phases while processing a page. The various stages, "Pre-page", "Input", "Pre-generation", "Header", and "Execution" are explained in the chapter "Key Concepts".

Action options are specified as name/value pairs as follows:

```
<action-option-name>=<action-option-value>
```

The value depends on the action option. For example:

```
.htd    text/html    EXECUTE[PREPAGE=PBE:
          preparePage; INPUT=PBE:input;
          PREGEN=PBE:generate]
          /BROWSER/HTML/NOBREAK
```

In this example, the action option PREPAGE has been set to 'PBE:preparePage', INPUT is set to 'PBE:input' and PREGEN is set to 'PBE:generate'.

The various action options are explained in detail below.

PREPAGE

PREPAGE specifies a callback to be called during the Pre-page stage. This is the first stage of processing a page request.

Specify the name of the callback function as the value of this option, as follows:

```
PREPAGE=<class/instance>:<callback-name>
```

- <class/instance> specifies the name of the class or object instance which defines the callback function. '<class/instance>:' is optional and need not be specified if the callback function has been declared globally. Class or instance method functions must be declared as public.
- <callback-name> is the name of the callback.

The prototype of the callback is as follows:

```
procedure <callback-name>(page, requestCookie, type)
    argument varchar page;
    argument integer requestCookie := $null;
    argument varchar type := "normal";
```

The 'page' parameter specifies the name and path of the requested page. 'request-eCookie' is a unique integer value which identifies the request browser. This integer value is issued to the browser by the application server. The 'type' parameter indicates the type of the page request, and can be one of the following:

- "normal" – This is a normal page request the came from the browser or other user agent.
- "goto" – This page request was initiated by the GOTO escape command.
- "error" – This page requests is the result of an error.

Note that in the prototype a PBT procedure declaration has been used in place of a normal method declaration. This is recommended for all callbacks in case the prototype is modified in future versions of the application server to include further parameters. If you use a procedure declaration additional parameters will be ignored, and the implementation will continue to work correctly.

INPUT

This INPUT action option specifies a callback to be used to input data from the Web. The callback function is called during the input phase once for each value received. Data is input in the form of name/value pairs. According to the HTTP standard, it is possible that the same name appears multiply times during input. This can be considered a list of values for that particular name.

Search args (arguments specified as part of the URL) are input before POST data. It is possible that a name is not specified for search args. In this case the application assigns a name of the form: search_arg<n>, where <n> is the position of the search arg in the argument list.

Specify the name of the callback function as the value of this option, as follows:

```
INPUT=<class/instance>:<callback-name>
```

- <class/instance> specifies the name of the class or object instance which defines the callback function. '<class/instance>:' is optional and need not be specified if the callback function has been declared globally. Class or instance method functions must be declared as public.
- <callback-name> is the name of the callback.

The INPUT callback has the following prototype:

```
procedure <callback-name>(inputName, inputValue, isFirst)
    argument varchar inputName;
    argument generic inputValue;
    argument boolean isFirst;
```

The 'inputName' parameter specifies the name of the input value. 'inputValue' is the value of the input. The application must be prepared to handle data of various types, from VARCHAR, VARBIN to LONGCHAR and LONGBIN values.

The 'isFirst' parameter specifies whether this is the first occurrence of 'inputName' or not. The first time 'inputName' occurs in the input phase 'isFirst' is set to \$true. All subsequence occurrences have 'isFirst' set to \$false.

NOTE: If no input callback is specified the application server creates a global variable for each input value in the same manner as earlier versions of the application server (pre 4.0). This method is no longer recommended because of the possibility of corrupting session code and data if the wrong value is set.

PREGEN

The PREGEN action option specifies a callback to be called during the Pre-page generation stage. This stage occurs immediately after input and just before action

page execution begins. The callback is generally used to process the data that has just been input.

The PREGEN callback function name in the action declaration is specified as follows as follows:

```
PREGEN=<class/instance>:<callback-name>
```

- '<class/instance>:' is optional, and may be used to specify the class or instance of the callback function.

The prototype of the PREGEN callback is as follows:

```
procedure <callback-name>()
```

No parameters have been specified for this callback.

HEADER

The HEADER action option specifies a callback that may be used to generate the HTTP header of the result page. Normally the HTTP header is generated automatically by the application server. Using this callback you can intercept the header which the application server has generated for a page and modify or make additions to it.

The HEADER callback function name in the action declaration is specified as follows as follows:

```
HEADER=<class/instance>:<callback-name>
```

- '<class/instance>:' is optional, and may be used to specify the class or instance of the callback function.

The prototype of the PREGEN callback is as follows:

```
procedure <callback-name>(page, header)
    argument varchar page;
    argument varchar header;
```

The 'page' parameter specifies the name and path of the requested page. The 'header' parameter specifies the HTTP header that would otherwise have been generated by the application server. The callback routine is expected to modify and then PRINT the header as required for the request page.

LIVETAGS

The LIVETAGS action option declares the name of a LiveTag map for all pages of the specified type. A LiveTag map is a file which contains a mapping between HTML/XML tags and method function calls. The LiveTags specified in the map will be used to process pages returned by this action.

How LiveTags work, and how to create a LiveTag map is explained in detail in the following chapter.

The LiveTag map is specified as follows:

```
LIVETAGS=<file-name>
```

- <file-name> is the name of the LiveTag map file. By default LiveTag map files are located in the application server 'Setup' folder.

Execution Options

Execution options specify additional information used for the processing of a requested page. There are a number of types of execution options. You may specify one option of each type in an action declaration. Defaults are used when an option for a particular type is not specified.

Session Type

This execution option specifies the type of session to be used to process the page. There are two types of sessions. Browser sessions are sessions associated with a particular browser or user agent. Each browser is allocated one browser session that remains "connected" to the browser until explicitly closed or the session times out. Pool sessions are not associated with a particular browser and are allocated for the duration of the request only.

By default, pages are processed by the browser session. The session type option specified in the action map can be overridden in a particular page by specifying BROWSER, POOL or NOBREAK as attribute of the <PBT> or <DAL> tag in the page itself.

- **BROWSER**

This option specifies that the page should be processed by the browser session. If the browser session is currently being used to process some other page, then it will be interrupted and an error returned to the browser. To prevent the execution of a page from being interrupted you can use the NOBREAK option explained below.

By default, execution of a page is interruptible. Pages that cannot be interrupted during processing will execute until generation of the result page is complete, or until the execution time-out. Execution time-out is a system parameter that can be set in the application server environment, and may also be set per session using the EXEC TIMEOUT escape command.

- **POOL**

This option specifies that the page should be processed by a pool session. If no pool sessions are available then the application server will create a new session and place this session in the pool. After processing the page pool sessions are returned to the pool where they can be allocated to subsequent requests. The execution of a page in a pool session cannot be interrupted. Execution time-out applies to pool session execution in the same manner as browser sessions.

- **NOBREAK**

This option specifies that the page should be processed by a browser session, and the execution of the page cannot be interrupted. In this case, subsequent page requests that are to be executed by the browser session are queued by the application server until the execution of the current page completes.

Page Type

This option specifies the type of the data of the result page. The default is HTML. Data placed in the result page comes from one of two different sources: static or dynamic. Static data comes from the static components in the source page. The application server does not modify static data in any way. Static data is copied directly to the output page.

Dynamic data is generated by the execution of embedded code and LiveTags. The application server processes the dynamic data according to the specified page type.

- **BINARY**

This option specifies that the page contains binary data. The application server is not concerned with the format of the binary data. As a result, dynamic data generated for pages that contain binary data is placed directly, without modification, in the result page.

A binary data page can be used to return graphical information or data specific to a browser plug-in. Java applets may also require data to be sent in a particular binary format.

- **TEXT**

Pages specified with the TEXT option consist of plain text. In this case the application server automatically converts all dynamic data generated to ASCII text. It also places a space character between each item dynamically gener-

ated by the session. A line feed character ('\n' - ASCII character 10) is generated at the end of a row of items.

- **HTML**
This options specifies that result page contains HTML data. HTML output is parsed by the application server in order to locate URLs in the result page. If URL IDs are being used for this page, then the application server will automatically add a search arg to the URL which contains the session ID of the browser. HTML entities are also substituted for characters in the range ASCII 128 and greater.
- **XML**
This options specifies that result page contains data in XML format.
- **STREAM**
The STREAM option specifies that data in the result page is formatted as an item stream of binary values. Each value includes a type, a length and the data. This data format is used by the PrimeBase JDBC driver when accessing the PrimeBase Database Server via the application server.

Proxy Caching

This option determines whether a result page may be cached by an internet proxy or not. The default is NOPROXY (caching not allowed). For pages using the RETURN and NOACCESS actions the default is caching allowed unless a callback option has been specified for the page.

- **NOPROXY**
Specify this option to prevent internet proxies from caching result pages.
- **PROXY**
This option indicates that data in the result page is largely static, and may be cached by internet proxies.

URL Browser ID

A browser ID identifies the browser to the application server. When required, PBAS will automatically add the browser ID as a search arg to URLs in HTML pages returned to the browser. This mechanism is an alternative to the using HTTP Cookies to identify a browser.

The application server does two things when placing a URL ID in a URL. Firstly, it adds a search arg to the URL in the form:

```
pb-id=<encrypted-browser-id>
```

Secondly, it adds a variable component to the search arg. This is in the form of two characters before the ID which change constantly. Constant modification of the URL is an additional mechanism used by the application server to prevent browsers and proxies from caching result pages.

In order to be sure result pages are not cached you should specify NOPROXY (see Proxy Caching above), and either the URLID or the VARIABLEURL option described below.

- **URLID**
This option places the browser ID in all URLs in the result page. It also adds a variable component to the URL to prevent browser and proxy caching.
- **FIXEDURLID**
This option places the browser ID in all URLs, but does not include the variable component. Result pages may be cached, but requests that reach the application server can be identified as coming from a particular browser.
- **NOURLID**
This option indicates that URLs in the result page should not be modified in any way.

- VARIABLEURL

This option modifies the URL by adding a dummy search arg. The browser ID is not placed in the URL and can therefore not be used to identify the browser. Browser identification, if required at all, must be done using HTTP cookies. This option is used to prevent browser and proxy caching of result pages.

As mentioned before, LiveTags™ and embedded code are the two methods used by the PrimeBase Application Server to create dynamic page content. LiveTags are preferred because they offer two major advantages:

- Firstly, LiveTags provide a clear separation between the user-interface (layout) and the program logic (code).
- Secondly, LiveTags facilitate the integration between Web pages and an object-oriented programming environment such as PrimeBase Enterprise Objects.

In this chapter we explain how LiveTags work and how to define new LiveTags. Note that the PBEO framework provides a complete implementation of LiveTags that provide all you need to implement a standard Web application. The information in this chapter will help you to extend this implementation or create there own for specialized purposes.

LiveTags are standard HTML or XML tags that are mapped to procedure or method function calls in the Web application. A procedure or method function used in a LiveTag map is known as a LiveTag handler function. The mapping from tags to handler functions is specified in a LiveTag map file.

In this way, any HTML or XML tags can be specified as "live". This includes standard tags and new tags defined by the programmer.

As explained in the chapter "Application Server Actions" (see page 32), a LiveTag map is determined for a particular source page type. The LiveTags specified in the LiveTag map may then be used in template documents associated with the given page type.

When the template document is loaded, the application server uses the LiveTag map to generate PBT source code for the LiveTags in the document. The source code generated is handed to the PrimeBase Virtual Machine for compilation.

The attributes of the HTML or XML LiveTag are passed as parameters to the LiveTag handler function.

As mentioned above, LiveTags are defined in a LiveTag map which, in turn, is specified in an action definition.

Each line of a LiveTag map file represents a mapping from an HTML or XML tag to a LiveTag handler function, and has the following format:

```
<tag-specification><tag-options><handler-function>' ('<handle-parameter>','...')
```

The meaning of each component of the LiveTag definition is as follows:

- <tag-specification> – The start or end tag specification.
- <tag-options> – The LiveTag options.
- <handler-function> – The name of the handler function.
- <handle-parameter> – Handler function parameters.

In other words: a LiveTag definition consists of a tag specification with various options followed by the handler function name and a number of handler function parameters in brackets, separated by commas.

8. LiveTags

What are LiveTags?

Defining LiveTags

Here is an example of a LiveTag definition:

```
</form> PBE:liveTagEnd("form", name)
```

In this example the tag specification is '</form>'. This declares the form end tag to be live. 'PBE:liveTagEnd' is the handler function. "form" and 'name' are handler parameters. "form" is a constant handler parameter and 'name' is an attribute parameter.

The various components of the LiveTag definition are explained in detail in the following sections.

Tag Specification

The tag specification must be given in the form of an HTML or XML tag. For example: '<FORM>', '<INPUT>', etc. To define an end tag as live, use the typical end tag syntax, for example: '</FORM>', '</SELECT>', etc. Note that if an end tag is specified as a LiveTag, then the corresponding start tag must also be defined as a LiveTag.

If both start tag and end tag are defined to be LiveTags this is known as a "tag block". Tag blocks are significant when calculating scope, and when used as control blocks (see page 43).

The tag specification may include a number of attributes. In this case the tag is only live if these attributes are present in the template document. For example, consider the following tag specification:

```
<INPUT MYLIVEINPUT>
```

This means that if an <INPUT> tag is encountered in the template document, the attribute MYLIVEINPUT is not present then the tag will be considered to be a static tag.

LiveTag Options

LiveTag options follow the tag specification in the LiveTag definition. LiveTag options can only be specified for start tag definitions. There are three possibilities:

- {
If the start tag is followed by a curly bracket, this indicates to the application server that this tag - and its associated end tag - are a "control block". Control blocks are used to create looping and branching LiveTags (see "Looping and Branching" on page 43).
- ...<
This character sequence indicates that the application server should include the HTML tag contents (up to the next start tag) as part of the LiveTag mapping. If this is done, then the contents can also be passed as one of the handler parameters to the handler function (see the description of ">...<" below). It is possible to specify this option along with the control block option described above. In this case the control block option must be specified first.
- ...<end-tag>
This indicates that the start tag, contents and end tag are all to be mapped to one handler function. The application server assumes that the contents do not contain HTML or XML tags (i.e. the contents are plain text). This is the case, for example, with HTML tags such as <TEXTAREA> and <SCRIPT>. If this sequence is used, then the contents of the HTML or XML tag are available as handler parameter (see the description of ">...<" below).

NOTE: This option may only be used when the end tag corresponding to this tag definition is not defined as a LiveTag.

Handler Functions

This is the name of a procedure, class method or instance method function. The name of the function must be fully qualified. The handler function must exist at run-time. If an instance method function is used, then the object instance must exist at run time.

This must be ensured by loading the procedure or creating the object instance during the initialization phase of the session.

Handler Function Parameters

Handler parameters specify value and order of the arguments passed to the handler function. A handler parameter may be one of the following:

- `<constant-value>`
A constant value, such as a number or some other PBT constant, or variable name.
- `<attribute-parameter>`
An attribute parameter indicates that the application server is to pass the value of a particular attribute as parameter to the handler function. The attribute parameter consists of an optional PBT data type followed by the attribute name followed by an optional scope specification. The scope specification must be prefixed by the '#' character. Attribute parameter scope is discussed in the section "Parameter Scope" below. The data type that is specified (if any) affects how the application server passes the attribute value to the handler function. How the various data types are handled is specified in the table below in the section "Parameter Types".
- `...`
This character sequence represents all other attributes not explicitly mentioned as an attribute parameter. The parameters are concatenated together as a single string value.
- `>...<`
This sequence represents the contents of the HTML or XML tag. The application server passes the contents of the tag as single parameter to the handler function, provided either the "...<" or ">...<" was used in the definition of the LiveTag.

Parameter Options

A number of options can be specified for attribute parameters. This is done by prefixing the attribute parameter with one of the following:

- `*`
If the attribute parameter is pre-fixed by a '*' then the attribute is required. This means that an error occurs if the attribute does not appear in tags in the template document.
- `!`
If the attribute parameter is pre-fixed by a '!' then the application server converts the attribute value to lower case before the call to the handler function is generated.
- `^`
If the attribute parameter is pre-fixed by a '^' then the application server converts the attribute value to upper case before the call to the handler function is generated.

Parameter Scope

A scope specification makes it possible to obtain attributes that have been specified for tag blocks that enclose the current tag. A tag block is defined by matching start and end LiveTags. Note that a tag block is required for scope references.

As mentioned before, the scope of a handler parameter can be appended to an

attribute parameter by using the "#" character. The scope of the tag itself is zero (#0). For example, this means that "name" is equivalent to "name#0". Scope one (#1) is the scope of the tag block immediately enclosing the tag. For example, an INPUT tag in a form can refer to the FORM tag attributes, which are a scope level one. In place of a number, scope may also be specified using a tag name of a tag in which the current block is nested. For example within an <OPTION> tag name#select refers to the name attribute of the <SELECT>-tag in which the <OPTION>-tag is nested.

Parameter Types

As explained above, an attribute parameter may include an optional type specification. The type specification helps the application server to determine how it should handle the value of the attribute. For example, if the type is indicated as BOOLEAN, then the application server will pass either \$true or \$false as argument to the procedure, depending on whether the attribute is present or not (i.e. for BOOLEAN type attributes, the value, if any, is ignored). On the other hand, a VARCHAR type parameter is always placed in quotes.

The type of the attribute parameter must be specified before the parameter options, for example:

```
varchar *!bgcolor
```

This specifies that the value of the attribute 'bgcolor' is to be passed as a parameter to the handler function. The bgcolor attribute has type VARCHAR. The options "*" and "!" indicate that the attribute bgcolor is required, and must be converted to lower case.

The following table specifies how the application server handles the various attribute types. The table shows how the value of the format of the attribute in the HTML/XML tag determines how the application server embeds the value in the call generated to the handler function, depending on the type of the value.

Type:	Attribute Format:			
	A=b	A="b"	A= or A	Attribute missing
No type specified	"b",	"b",	"" ,	,
VARCHAR	"b",	"b",	"" ,	,
BOOLEAN	\$true,	\$true,	\$true,	\$false,
DATE/TIME	"b",	"b",	\$null,	,
MONEY/DECIMAL	b,	"b",	"" ,	,
INTEGER/FLOAT	b,	"b",	\$null,	,
VARBIN	b,	"b",	"" ,	,
OBJNAME	"b",	"b",	,	,
CURSOR	b,	b,	,	,

In some cases, indicated in the table by a "," on its own, the application server leaves the parameter missing from the call. In these cases, the parameter will take on the default value for the parameter as indicated by the handler function.

NOTE: The handler function must be defined as a PBT procedure in order to use this feature. PBT method and function definitions do not allow the specification of default value for a missing input parameter.

PBT Attribute Type

The special attribute type PBT indicates that the value of the attribute is a PBT expression. In this case the application server strips any quotes surrounding the attribute value, and places the result directly in the code generated to call the handler function. In this way the expression is evaluated when the handler function call is executed. The result of the expression evaluation is passed as the input parameter to the handler function.

Using various escape commands it is possible create LiveTags that loop and branch within the template document. Two features of LiveTags are important for creating looping and branching tags: LiveTag Control Blocks and LiveTag Escape Commands.

Looping and Branching

LiveTag Control Blocks

LiveTag Control Blocks are tag blocks that have been specified with the '{' option. For example:

```
<if>{    PBE:ifTag($true, pbt cond)
</if>    PBE:ifTag($false)
```

In this definition, the `<if>` tag is declared using the '{'. This declares the `<if>` tag as a control block. Note that the `</if>` (end `<if>` tag) must also be declared as a LiveTag. This is required for declaring a tag block (used for scoping). In other words, a control block is specialization of tag block.

While processing a template document, the start and end tags of a control block serve as reference points for the LiveTag Escape Commands.

LiveTag Escape Commands

Using the LiveTag escape commands, the application server can be directed to continue processing of the template document at a different point. The start and end tags of the current control block server as reference points these commands.

There are four LiveTag Escape commands:

- **REPEAT**
This command tells the application server to continue processing the template document at the start tag of the current control block. In this case, the LiveTag handler function of the start tag will be called again.
- **LOOP**
This command tells the application server to continue processing the template document at the point immediately after the start tag of the current control block.
- **CONTINUE**
The application server is instructed to continue processing the template document at the end tag of the current control block. The LiveTag handler function of the end tag will be called again.
- **BREAK**
Exit the current control block. The application server will continue processing the template document at the point immediately following the end tag of the control block.

Using various LiveTag escape commands, the handler functions of the LiveTags of a control block can implement the required looping and branching functionality.

APPENDIX

A. Escape Commands

In general, escape commands are used to instruct the PrimeBase Application Server to perform a particular action.

The varbin \$null value is an "escape item". When PBAS encounters a NULL value of type VARBIN, it assumes that a command will follow. After the command comes a variable number of arguments (depending on the command). This section describes the escape commands currently supported by the application server.

Session / Execution Control

- print/2 (varbin) \$null, "END";
Closes the User/Browser Session. Use this command to do an explicit "logout" where possible.
- print/2 (varbin) \$null, "END ALL";
Closes all sessions.
- print/2 (varbin) \$null, "QUIT";
This command terminates execution of the current page, and returns the result page to the browser as generated up to that point. Be careful not to generate incomplete HTML pages using this command. Note that this command has no arguments.
- print/2 (varbin) \$null, "REDIRECT", <url-for-redirection>;
Terminate execution of the current page, and send a URL redirect command to the browser. The browser is directed to the URL specified by <url-for-redirection>.
- print/2 (varbin) \$null, "EXEC/EXECUTE", <program-text>
Use this command to accumulate program text which will be executed by the application server when the current embedded code fragment or LiveTag has completed execution. This command is useful for database servers that do not fully support the execution of strings (EXECUTE and EXECUTE FROM).
- print/2 (varbin) \$null, "GOTO", <page>;
This command transfers control immediately to another page. The application server discards output for the current page, and starts execution of the new page. Execution continues in the same session. <page> is the name and path (relative to the docs folder of the application server) of the page to be executed. The page path and name must be given in UNIX file path notation.
- print/2 (varbin) \$null, "ACTION", <action-definition>;
This escape command is used to define an action. Action declared in this manner either override or extend the actions already defined for the application server.
The <action-definition> is a string value with the same format as the action definitions in the Action Map file (see the chapter "Application Server Actions" on page 32).
- print/2 (varbin) \$null, "RELOAD CONVCHAR";
This escape command reloads the 'convchar.in' and the 'convchar.out' files. These files specify a mapping of from the character set used by the Web browser and the character set used by the application server.
- print/2 (varbin) \$null, "SYSTEM PATH", <path>;
This escape command allows you to set a search path for system files, such as LiveTag map files. This location will then be searched in addition to the 'Setup' folder. The path must be specified in the native file system format. Set the path to empty ("") to revert to the standard search path.

Set Time-outs Locally

It is possible to set time-out values locally in the session. The global time-outs set

using the system parameters are the defaults.

- `print/2 (varbin) $null, "EXEC/EXECUTE TIMEOUT", <timeout-in-seconds>;`
Override the default "Execution Timeout" (PBAS environment parameter 2705) for the current session only.
- `print/2 (varbin) $null, "IDLE TIMEOUT", <timeout-in-seconds>;`
Override the default "Idle Time Before Closing" (PBAS environment parameter 2702) for the current session only.
- `print/2 (varbin) $null, "ERROR TIMEOUT", <timeout-in-seconds>;`
Override the default "Close Error Timeout" (PBAS environment parameter 2704) for the current session only.
- `print/2 (varbin) $null, "RECYCLE TIMEOUT", <timeout-in-seconds>;`
Override the default "Idle Time Before Re-cycle" (PBAS environment parameter 2703) for the current session only.

The following escape commands print to both the log, and the console, depending on the print or logging level:

- `print/2 (varbin) $null, "ERROR", <error-message>;`
The command sets the error status. Execution of the page terminates, and the error is returned to the user. A message printed or logged when the corresponding print/logging level is higher or equal to "Error".
- `print/2 (varbin) $null, "PROTOCOL", <protocol-message>;`
This command allows you to generate a protocol message just like the messages generated by the application server itself. In this way you can add custom entries to the protocol log. The message is printed or logged when the corresponding print/logging level is higher or equal to "Error".
- `print/2 (varbin) $null, "DEBUG", <debug-message>;`
Prints/logs the message if the print/logging level is higher or equal to "Warning". You may use this command to debug your PBT sub-programs as they are executed by the application server.
- `print/2 (varbin) $null, "COMMENT", <comment-message>;`
Prints/logs the message when the print/logging level is higher or equal to "Trace".

Certain parameters affecting the handling of binary data by the application server may be set locally in a session.

- `print/2 (varbin) $null, "BLOB FILE NAME", <pattern-for-temp-files>;`
Set the pattern of file names to be used when creating temporary files generated by printing binary values. The wildcard "*" must be used, and is replaced by a unique ID generated by the application server.
For example "tmp*.wav", causes the application server to generate temporary file names for sound values loaded from the database. This escape command overrides the system parameter 2728 "Blob File Name" for the current session only.
- `print/2 (varbin) $null, "BLOB URL", <pattern-for-url>;`
Set the URL tag generated around the temporary file URL (set using system parameter 2739 "Temporary File URL") when a binary value is printed. The wildcard "*" must be used to indicate where the file name is to be placed.
For example, if a binary value is printed, and the binary value represents an image, then set the pattern `` will cause the browser to display the picture correctly.
Binary values can be printed without any need to print such a tag around it. This escape command overrides the system parameter 2730 "Blob URL Tag" for the current session only.
- `print/2 (varbin) $null, "RETURN", <content-type>, <unix-style-file-name>;`

Errors, Debugs, Protocols and Comments

Binary Value Control

This escape command stops execution of the page, and returns the binary data in the file specified. The file name is relative to the HTML Page Location directory of the application server, unless an absolute path is specified.

Set/Get/Callback Variable Escapes

The SET/GET/CALLBACK Variable Escapes cause the application server to assign a certain variable with the appropriate value, before the next embedded code fragment or LiveTag is executed.

- `print/2 (varbin) $null, "CALLBACK", <function-prototype>;`
The escape commands sets up a callback that will be called before the next code fragment is executed. `<function-prototype>` is a string with the following format:

```
<function-name> ( <variable-name> , ... )
```

This means that the function prototype consists of a function name followed by any number of system variables names in parenthesis. System variables are described below.

- `print/2 (varbin) $null, "GET", <variable-name>;`
The GET escape command retrieves the value of a system variable from the application server. `<variable-name>` is a string containing the name of the system variable. The calling code must use `$input()` to receive the value immediately after issuing the escape command. For example:

```
print/2 (varbin) $null, "GET", "user_agent";
varchar the_user_agent := $input();
```

- `print/2 (varbin) $null, "SET", <variable-name>;`
The SET escape command retrieves the value of a system variable from the application server. `<variable-name>` is a string containing the name of the system variable. The application server creates a global variable with the same name as the system variable and sets the global variable to the current value. The global variable is available in the following code fragment.

The following is a complete list of the system variables that may be used in the GET, SET and CALLBACK escape commands.

- `user_agent`
This is a string which identifies the user agent (browser) connected to the current session.
- `remote_host`
This is the IP address or host name of the machine on which the user agent is running.
- `agent_sessions`
This is the number of sessions currently open and in use by various user agents.
- `http_referer`
If provided by the user agent, then this is the URL of the page that "initiated" the current page request.
- `http_server_protocol`
This is the HTTP level protocol used to send the current request. For example, HTTP or SHTTP.
- `http_server_name`
This is the IP address or host name of the local machine.
- `http_server_port`
This is the port used to make the current page request.
- `http_script_name`
This is the script name of the current page request. The script name is the component of the request URL that refers to the application server. This value is

empty if the application server is accessed directly using the build-in HTTP server.

- `http_path_info`
This is the path information of the current request. This is the component of the URL that follows the script name. It refers to a document in the application server 'Docs' folder.
- `http_content_type`
This is the MIME content type of the current page request.
- `http_cookie`
This is the value of the HTTP Cookie header of the current request.
- `http_post_data`
This variable contains the entire body of the current request.
- `http_request_method`
The method of the current request for example, POST, GET, HEAD, etc.
- `http_query_string`
This variable contains the string following the first '?' character in the request URL.
- `primebase_version`
This is a string value describing the version of the application server.
- `current_page`
This is the path information of the page currently being executed. This may be different from the 'http_path_info' if the page was invoked using the GOTO escape command.
- `platform_string`
This variable contains information about the platform on which the application server is running.
- `pbas_cookie`
This is a 4-byte integer value that uniquely identifies the user agent.
- `request_cookie`
This is the PBAS cookie value (user agent ID) that was sent with the current request. If the value is zero then this means that the user agent did not send an ID with the request. This can be considered a "new comer" to the Web application. If the value differs from the value of 'pbas_cookie' then the user agent sent a "stale cookie" with the current request. A stale cookie is a user agent ID associated with a session that has been closed or has timed out.
- `pbas_id`
This is the ID of the application server that issued the PBAS cookie value.
- `pbas_cookie_name`
This is a string value that is used to identify the PBAS cookie in a URL. Use this value in combination with the 'pbas_cookie_string' to place the PBAS cookie in a URL.
- `pbas_cookie_string`
This is a string value that represents the PBAS cookie value in a URL. This is an encrypted version of the PBAS cookie and is known as the URL ID. Use this value in combination with the 'pbas_cookie_name' to place the PBAS cookie in a URL.

LiveTag escapes may be used within a LiveTag control block to perform flow control during the execution of an HTML page. Blocks can be skipped or repeated using the escapes described here.

- `print/2 (varbin) $null, "BREAK";`
Continue execution of the HTML page after the end of the current LiveTag control block. This escape may be used to exit a loop, or skip a control block if a condition is not met.

LiveTag Escapes

- `print/2 (varbin) $null, "CONTINUE";`
Jump to the end tag of the current LiveTag control block.
- `print/2 (varbin) $null, "REPEAT";`
Jump to the begin tag in the current control block. REPEAT can be used to continue a loop which checks the loop condition in the start tag.
- `print/2 (varbin) $null, "LOOP";`
Continue execution at the next point immediately after the begin tag of the current control block. This escape can be used to create a loop which uses the start tag to initialize the loop and then checks the loop condition in the end tag.

Background Processing

- `print/2 (varbin) $null, "REPLY", <process-name>;`
The REPLY escape commands sends the reply page as generated so to the user agent. Execution of the page continues in the background. Further output is not sent to the user agent. <process-name> is the name given to the background process.
- `print/2 (varbin) $null, "NAME", <process-name>;`
Set the process name of the current background process.
- `print/2 (varbin) $null, "OUTPUT", <process-name> <variable>;`
Retrieve the output of a particular background process. <variable> is the name of a global variable that will be set with output of the given process.
- `print/2 (varbin) $null, "SET OUTPUT", <message>;`
Set the output of the current background process to the given text.
- `print/2 (varbin) $null, "KILL", <process-name>;`
This escape command terminates the execution of a particular background process.
- `print/2 (varbin) $null, "STATUS", <process-name>;`
Retrieve the status of a particular background process. The status is placed in a global variable with the same name as the process.
Status may be one of the following:
 - "" (empty string) – This means no process exists with the given name.
 - "running" – The process is executing.
 - "zombie" – The process has terminated, and is waiting to be killed.
- `print/2 (varbin) $null, "SEND", <message>;`
Add the given message to the output of the current background process.

NOTE: A pool session should be used to execute a background process. Using a browser request would block any further requests from the user agent.

B. System Parameters

System parameters are stored in the PBAS environment file, called 'pbas.env', which is located in the 'Setup' folder (see page 28). You can use the "System Web Interface" (see page 16) or the program "EnvironmentEdit" on the Mac, 'pbee.exe' under Windows or 'pbee' under UNIX/Linux to alter system parameters.

General Configuration

- **300** - HTML Page Location
This path the application server searches for pages requested. All paths are relative to this path. For security reasons, always set this parameter.
- **302** - Server Name
This is the name the primebase.exe (the component started by the Web server under Windows and UNIX/Linux) uses to access the application server. Set this parameter if you are running more than one instance of the application server. Make sure that the primebase.exe can access a copy (or the same) of the

pbas.env file.

- **2700** - Default Page
Use this parameter to set the name of the default page, returned by the application server when no page is explicitly requested. The default is "index.htm".
- **2760** - File Extensions
Use this parameter to determine which types of files will be returned by the application server. If set to an empty string (default), the application server will return all types of files. Possible value is "htm, html, htm, lml" to restrict the range of file types.

The following configuration parameters effect the creation of new sessions:

- **308** - Maximum Open Sessions
This is the maximum number of sessions that will be opened by the application server. It will try to close sessions already in-use to avoid exceeding this limit (see re-cycle timeout parameter above). If the application server cannot close an existing session, an error message is returned to the client.

NOTE: You should not set the value greater than the number of connections that will be accepted by your database server.

- **2701** - Open Sessions Ready
Opening and initializing a new session can take some time. For this reason, the application server can be configured to keep a certain number of new sessions ready for browsers that have not yet been allocated a session.
- **2706** - Default Connection Alias
Normally, the application server creates sessions using the default connection definition in the "connect.def" file (the top-most entry in the file). If you set this variable to a non-blank value, then the application server will use the connection definition with the given alias name instead.

- **2702** - Idle Time Before Closing
This is the time in seconds a session may be idle before it is closed by the application server. Set the value to zero if the session should never timeout.
- **2703** - Idle Time Before Re-cycle
If the maximum number of sessions has been opened, the application server will try to re-cycle sessions currently in use. For this purpose the application server selects the least-recently used session. If this session has been idle for the specified time, then it is re-cycled. Set this value greater than Idle Time Before Closing to prevent re-cycling, or to -1 if sessions should never be recycled. Set the value to zero if sessions should be recycled as soon as the maximum is exceeded.

NOTE: If you set the 'Idle Time Before Re-cycle' to -1 be sure that the 'Idle Time Before Closing' is not zero, or the application server will never be able to close a session!

- **2704** - Close Error Timeout
This parameter specifies in seconds how long an error message is stored for a browser whose session has been closed due to a recycle, idle timeout or some other reason. This error warns users that their session has been closed and any associated state information has been lost. If you do not want any error to appear, set this value to zero.

Session Creation & Configuration

Session Time-outs

Protocol Logging / Display

The application server creates a protocol file for logging errors, warning, etc. Messages are also displaced on the console.

- **158** - Log File Creator
This parameter is only significant on the Mac. It determines the "Creator" of the log file (type is always "TEXT"). The default is "txt", which means SimpleText is the creator.
- **340** - Protocol Log File
The name of the protocol log file. The default is "pbas.log". The protocol log file is always placed in the local setup folder.
- **342** - Protocol Log Level
The protocol log level determines which messages are written to the log. The levels are as follows:

ID	Level
0	OFF - Lowest level
1	ERRORS
2	WARNINGS
3	TRACE - Highest Level

Messages are logged if the level of the message is equal to, or less than the level set here. For example is level is set to "Warnings", then both errors and warning are logged. The default protocol log level is "ERRORS". At this level a "server started", and "server shutdown" message is written to the log, and all errors are recorded in the log.

Note that the log file can be removed, renamed or deleted at any time while the application server is running.

- **343** - Protocol Display Level
This is the protocol level for the console. It determines which messages are printed to the console. Levels are set the same as for the protocol log above. The default protocol display level is "TRACE".

Identification of Browsers

These system parameters control how the application server identifies the browser/user of an incoming request.

- **2707** - Use Client Address
Indicates whether the application server should use the client IP address to determine the origin of a request. Note that this is not a reliable way to identify the user, because the IP address is usually that of the proxy server of the user. This setting is off by default.
- **2708** - PBAS URL Pattern
If not blank the application server will place its cookie value for a particular user in URLs matching the given pattern. This pattern should describe URLs that refer to the application server. Wildcards "*" (any character sequence), and "?" (any single character) may be used. Cookies are placed in URLs to back-up the normal cookie mechanism, which does not always work. The default value is "*.htd", which means that the application server recognizes URLs referencing files ending with ".htd" as references to itself.
- **2709** - Use HTTP Cookies
This parameter determines whether the application server uses HTTP cookies to store its own cookie value in the user's browser. This is the most reliable method of identifying where a request comes from, and is on by default. If you turn this parameter off, then you must ensure that URL Pattern of the application server is set correctly, otherwise it will have no good way to determine the origin of a request.

- **2740** - Cookie Seed
A random value used to seed the cookie sequence generated by your site. Set this value once when the site is taken into production. The random value set here prevents users from guessing valid cookie values for your site.
- **2741** - Session Number
The next session number to be issued by the application server. This value is encrypted before being returned as a cookie to the user. Set this value once when your site is installed. A random starting point for the session number ensures users cannot guess valid session numbers.

NOTE: *Resetting this value will cause the application server to generate cookie values it has already issued. This could cause some problems with browsers using old cookies, and in particular, will cause sessions to become mixed up if you are storing cookie values in a database.*

- **2720** - Temporary Directory Name
The name of the directory in which temporary file are placed.
- **2721** - Temporary Directory Path
The location of the temporary directory.

BLOB Files & URL Parameters

NOTE: *The temporary directory must be accessible by the Web server.*

- **2723** - Blob Disk Cache Size
The amount of space, in bytes, that may be used to store temporary files.
- **2725** - Cache Blob Timeout
The timeout is to indicate whether the application server can delete a temporary file or not. When a binary value is returned to the browser, the session has a lock on the associated binary file. When the session requests another page or is closed, the lock is removed. However, it may be some time before either of these events occur. In this case, the Cache Blob Timeout value is used to determine whether the lock can be removed.
- **2728** - Blob File Name
Blob files are generated using this pattern. The pattern must contain a "*" which is replaced by the BLOB number by the application server. For example, if you are generating GIF files, then "tmp*.gif" is a good pattern (this is also the default). This variable may also be set locally in a session using the BLOB FILE NAME escape command, for example:

```
print varbin $null, 'BLOB FILE NAME', 'tmp*.jpeg';
```

- **2729** - Temporary File URL
When a binary value is printed, the application server generates a temporary file and places the file name in the HTML page. You can use the parameter to place the file name in the context of a URL.
The value must contain a "*". The "*" is replaced by the name of the temporary file (as determined by parameter 2728 above). Set this parameter is such a way that the Web server can find the temporary file generated by the application server. This value will depend on where the application server places temporary files as set using parameter 2720 and 2721. The essential difference is that 2720 and 2721 determine the location of temporary files from the point of view of the application server, and this parameter determines the location of temporary files from the Web server's point of view.
- **2730** - Blob URL Tag
When you print a binary value, the application server can not only place the URL of the temporary file in the HTML page, but can also place this file name in

the context of an HTML tag.

For example, if you are printing binary values that are actually pictures, then you can set this parameter to: `` (this is also the default for this parameter). The application server replaces the `***` with the URL of the temporary file (as determined by parameter 2729 above).

List Variable Pattern

- **2750** - List Variable Pattern
Identifies variables to be treated as list type variables. A list variable is a PBT variable which may contain a number of values separated by comma. List variables have type VARCHAR. Normally when a variable is set by HTML form (or search args) multiple assignments overwrite each other. In the case of a list variable, the application server concatenates the values separating them by a comma.

Activation Parameters

- **303** - Serial Number
The serial number issued to you on registration. Demonstration serial numbers begin with "DV".
- **304** - Registration Name
The name of the person or company to whom the PrimeBase Application Server has been registered.

NOTE: We require a registration name before we can issue an activation key.

- **307** - Registered Maximum Connections
This parameter should be set to 0 (zero), which means unlimited. This is the registered maximum number of sessions your copy of the application server is licensed to open. Normally an unlimited licence is issued.
- **314** - Activation Key
This is a value of the form "XXXX-XXXX-XXXX". It should be set to the activation key value issued to you on registration. The activation key verifies the settings of all activation parameters. If verification fails, the application server will return a message that it has expired.
If you have a valid activation key, but you still get the expired message, then check all activation system parameters carefully. Values are case-sensitive in all but the registration name.
- **315** - Expiry Date
Demonstration activation keys all have an expiry date. If you have a demo version of the application server, the expiry date is stored here. It is a string of the form: MM/YYYY. The application server always expires on the first day of the month specified. If you are a registered user, your server will have no expiry date. In this case, set this parameter to 0 (zero).

HTTP Controls

- **2500** - HTTP Port Number
HTTP Port to use when using the built-in web server. To use the port number specified here, set parameters 2501 (described below) to 1.
- **2501** - HTTP Usage
Use this variable to determine the application server's Web publishing behavior. If set to 0, then the application server uses the server name (parameter 302) to generate a port number. If 1, then the port number specified by parameter 2500 (HTTP Port Number) is used. If set to 2 then the application server will not publish an HTTP port. In this case, the application server can only be accessed using the standard CGI mechanism.

- **2712** - Browser Error Page

An error occurring during execution of a page in a browser session will automatically cause a GOTO (see GOTO escape command, page 44) to this page. Information about the error that occurred is placed in global variables as specified in "Error Page Settings" on page 53.

This capability allows you to format the error messages returned to the browser (or otherwise handle the error) when a runtime error occurs. If no page is specified, the application server will return the standard error message page.

- **2713** - Pool Error Page

This page is executed if an error occurs during the execution of a page in a pool session. The error page is executed in the same session as the session in which the error just occurred. Global variables are set as described above for the "Browser Error Page". Pool Error Page and Browser Error Page may be identical.

Error Page Settings

The following is a list of errors generated by the PrimeBase Application Server:

Code	Message
-19001	Page execution interrupted (*)
-19002	Application Server has an internal error
-10112	Insufficient memory to perform operation (*)
-19003	No file specified in URL
-19004	The Application Server has shutdown (*).
-19005	Session closed because the session slot was re-cycled (*)
-19006	Session closed due to idle timeout (*)
-19007	Session closed due to execution timeout (*)
-19008	Session closed due to user logout(*)
-19009	Session deleted
-19010	I/O error on 'connect.def' file
-19011	'connect.def' file contains no entries
-19012	Alias not found in 'connect.def' file
-19013	Currently no session available (*)
-19014	Privilege violation (*)
-19015	Invalid plug-in
-19016	Error opening file
-19017	Error reading file
-19018	Error creating process
-19019	Session closed due to Admin login
-19021	Feature not implemented

(*) These errors are most important, and may occur during normal operation. Other errors indicate a configuration or programming error.

C. Application Server Errors

The following errors can occur during the execution of pages:

Code	Message
-19022	Escape command expected
-19023	Invalid escape command
-19024	Escape command argument
-19025	Invalid escape command argument
-19026	Incorrect type for escape command argument

Contact us, or check out our Web site for further information, pricing and availability of the latest release of the PrimeBase™ Application Server.

e-mail: info@primebase.com

www: <http://www.primebase.com>

© 2001-2008, PrimeBase Systems GmbH. All rights reserved.