Introduction to the Web Request Broker™

Release 2.1

Part No. [[Part Number]]

ORACLE®

Enabling the Information Age

Introduction to the Web Request Broker, 2.1

Part No. [[Part Number]]

Copyright © Oracle Corporation 1996

All rights reserved. Printed in the U.S.A.

Primary Author: Martin Gruber, Kennan Rossi

Contributing Authors: Charles Prael

Contributors: Seshu Adunuthula, Mala Anand, Thomas Kyte, Shehzaad Nakhoda

Preface

Introduction to the Web Request Broker is part of the Oracle WebServer documentation set, which contains:

- This book, *Introduction to the Web Request Broker*
- I*ntroduction to the Oracle WebServer*
- The *Oracle WebServer Installation Guide* for your platform.
- The Oracle WebServer online documentation

This book is a high-level overview of the Web Request Broker (WRB), providing conceptual background information for the online documentation, which provides in-depth information in `html` format. To view the `html` files, you can use any standard web browser program that supports tables and client-side image maps.

Once you have installed the Oracle WebServer, you can follow the link from the product home page to the online documentation.

This Preface discusses this Guide's:

- Organization
- Typographic conventions
- Related documents

How this Guide is Organized

- Chapter 1, "Overview of The Web Request Broker (WRB)" describes the Web Request Broker architecture and features.

- Chapter 2, “The PL / SQL Agent” briefly describes the features of the PL / SQL Agent WRB cartridge.
- Chapter 3, “The Java Cartridge” briefly describes the features of the Java WRB cartridge.
- Chapter 4, “LiveHTML” briefly describes the features of the LiveHTML WRB cartridge.
- Chapter 5, “Web Request Broker Administration”provides a brief introduction to WRB administration using the WebServer Manager, a collection of HTML forms and related utilities.

Conventions Used in This Manual

Feature	**Example**	**Explanation**
monospace	`enum`	Identifies code elements.
boldface	**mna.h** **timeout**	Identifies file names and function arguments when used in text.
italics	*file1*	Identifies a place holder in command or function call syntax; replace this place holder with a specific value or string.
ellipses	n,...	Indicates that the preceding item can be repeated any number of times.

Table 1: Conventions

Example Conventions

This Guide shows code in this font:

```
applet.addParam("text", "This is an applet test.");
```

Related Documents

Part No.	Document Title
	Introduction to the Oracle WebServer

Table 2: Related Documents

Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of our printed manuals is a Reader's Comment Form, which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form is not available, please use the following address or FAX number.

Oracle WebServer Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
FAX: 415-506-7200
email: owsdoc@us.oracle.com

Contents

1 Overview of The Web Request Broker (WRB)

The Web Request Broker is the central component of the Oracle WebServer. It is an asynchronous request handler with an API (Application Program Interface) that enables it to interface dynamically and seamlessly to various back-end technologies called “WRB Cartridges”. It provides an architecture that allows server-side web applications to run under any HTTP server to which the WRB has been ported.

The Web Request Broker offers capabilities similar to CGI scripting. In fact, you can use the WRB API to access CGI data sent by the client. WRB applications, called *cartridges*, however, take advantage of the WRB’s multi-process architecture to get much better performance than ordinary CGI scripts.

The WRB architecture also makes WRB cartridges extremely scalable—that is, they can handle small request loads economically and large request loads efficiently.

Whenever the Web Listener receives a URL that calls for the WRB, it passes execution of the request to the *WRB Dispatcher* (or simply *Dispatcher)*. The Dispatcher determines which cartridge the request is for and passes execution to it. The result is that the Listener can receive and validate URLs coming in, while each request is handed off to a process that executes it in the background.

WRB cartridges can be of the following types:

- The PL/SQL Agent. This cartridge executes PL/SQL commands stored in the database. It is better optimized for database access than the Java cartridge, but doesn’t have all of Java’s functionality.
- The Java Cartridge. This cartridge lets you execute Java on the server to generate dynamic Web pages. You can also execute PL/SQL from within Java using this cartridge.

- LiveHTML. This cartridge is Oracle's implementation and extension of the industry-standard Server Side Includes functionality. *LiveHTML* enables you to include in your Web pages the output of any program that your Operating System can execute.
- Your Own Cartridges. Since the WRB uses an open API, you can write your own cartridges to use it. Currently, you use the C language to write WRB cartridges. In the future, however, the WRB API will also be available in other languages. Instruction and reference on writing WRB cartridges is available in application development section of the online documentation.
- Third-Party Cartridges. Also since the WRB uses an open API, various independent vendors are writing cartridges for it.

WRB Architecture

The WRB consists of these components:

- The WRB Dispatcher
- WRB cartridges
- The WRB application engine
- WRB execution instances (WRBXs)

A WRB cartridge is implemented as a shared library that uses the WRB API to handle HTTP requests from web clients.

The *WRB Dispatcher* is a program that provides the interface between Web Listeners and WRB cartridges. When the Web Listener receives an HTTP request directed to a WRB cartridge, it forwards the request to the WRB Dispatcher. The Dispatcher determines which WRB cartridge to send a given request to on the basis of the path and the file extension (MIME type) specified in the URL. You configure the WRB in the WebServer Manager to determine which combinations of path and extension correspond to which cartridges.

The Dispatcher chooses a WRBX (WRB execution instance) of the cartridge to handle the request. For each WRB cartridge, there are varying-number of WRBXs, which are processes created and destroyed as needed to handle the workload. A minimum number of WRBXs are kept running continuously, so that most requests will not require that a new process be spawned.

The WRBX is composed of two parts: a copy of the WRB application engine, and the WRB cartridge shared library.

The *WRB application engine* is the executable program that implements the WRB API. It provides the interface between WRB cartridges and the WRB Dispatcher, directs WRB cartridge flow of control, and provides services for WRB cartridges to use.

The WRB architecture is shown in Figure 1-1.

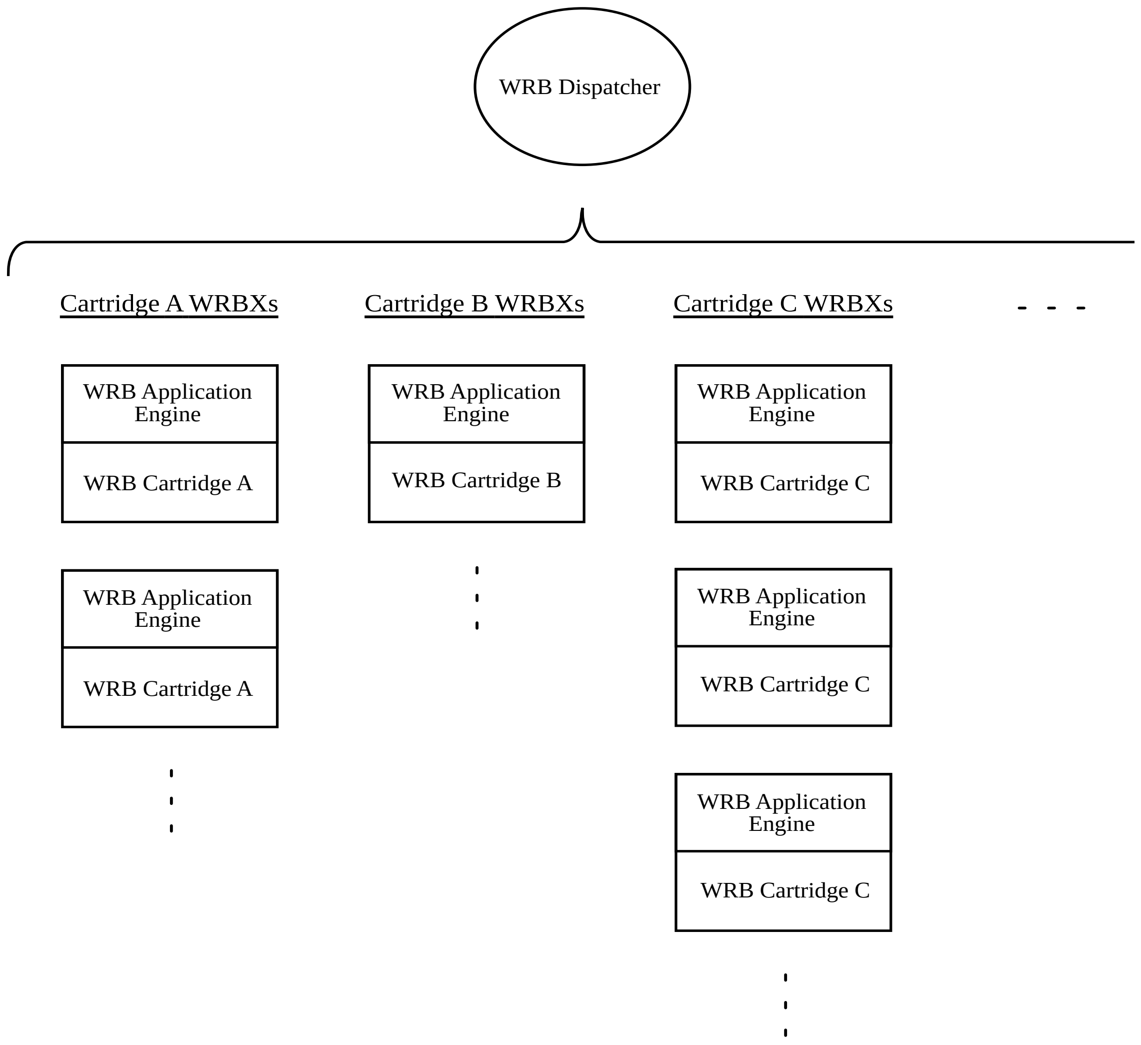

Figure 1-1: The WRB Dispatcher manages execution instances of WRB cartridges

The WRB configuration data specifies for each cartridge the maximum number of WRBXs that may run at once, and the minimum number of WRBXs that must always be running. You can tune your WRB performance by adjusting these values using the Web Request Broker administration pages).‘

The WRB API

The WRB API defines:

- Cartridge functions—the interfaces the WRB application engine uses to call your application.
- WRB application engine functions—the interfaces the WRB application engine provides for your application to use.

To make your cartridge functions available to the WRB application engine, you store pointers to the cartridge functions your application defines in a function table (see The Entry-Point Function).

The WRB application engine functions are declared in the header **wrb.h**. The function names all begin with “`WRB`.”

WRB Cartridge Security

WRB cartridges can augment the access control specified by the Listener with cartridge-specific control. The cartridges included with the WebServer do not do this, but the WRB API provides a callback routine that enables you to put this functionality in cartridges that you write. This routine enables your cartridge to do any of the following:

- Prompt for a username and password, using either a Basic or a Digest scheme. The difference between Basic and Digest is that a Digest scheme encrypts the password (for more information on these terms, see *Introduction to Oracle WebServer*).
- Specify by name some restriction scheme to be applied to this request. Such restriction schemes are defined in the Listener and limit access by IP address or Domain Name.

For more information on how WRB cartridges work and how to write your own, refer to the WRB specification that accompanies the WebServer online documentation.

CHAPTER

2 The PL / SQL Agent

The PL / SQL Agent can be invoked through either the WRB or CGI, as determined by the directory and MIME type mappings set by the WebServer Administrator. If it is through CGI, the script name specified in the URL must be "owa". If it is through the WRB, the directory and MIME type mappings are sufficient to specify the PL / SQL Agent. The PL / SQL Agent executes application code written in PL / SQL and returns the output in HTML form for the Web Listener to output as a Web page.

PL / SQL procedures are stored in the database. The PL / SQL Agent invokes them by issuing commands to the database, which then performs the actual execution and sends the output and status messages back to the PL / SQL Agent.

Since PL / SQL is actually executed in the database, the PL / SQL Agent, whether executed through the WRB or CGI, must connect to the Oracle7 Server to run. If the PL / SQL Agent uses the WRB, the WRBX connects to the database in the following two stages:

- It establishes the connection. The WRBX does this as soon as it is created.
- It logs on to the database. This is a separate operation, and the WRBX does not do this until a request comes in, so as to give each request a separate database session. Once the request is handled, the WRBX logs off of the database, but keeps the connection intact.

Since the first stage actually takes most of the time involved in establishing a database session, this technique speeds execution considerably, yet the effect is as though each request connected to the database independently. If run through CGI, the PL / SQL Agent must go through the entire procedure for each request. The username, schema, and password that the PL / SQL Agent uses to connect is specified by the URL through the use of a DCD, as explained below.

Specifying the Database Connection

A URL that invokes the PL / SQL Agent must specify a DCD (Database Connection Descriptor). This is an OS file maintained by the WebServer that provides the username, password, database, and other information to be used to establish the database connection. The DCD determines both the database access privileges the PL / SQL Agent has when executing this request and the schema (portion of the database) that it accesses. The filename of the DCD is given in the URL as a file within the directory configured for the PL / SQL Agent. The file need not actually reside in that directory; its association with the PL / SQL Agent is set in the WebServer Manager PL / SQL Agent configuration page. If the PL / SQL Agent is invoked through CGI, the DCD precedes the script name “owa”, with the two separated by a slash (/).

Oracle WebServer also provides you with Java classes that can invoke PL / SQL. For more information, see Java or the Java Interpreter.

The PL/SQL WebToolkit

To make it easier for you to develop Web applications using Oracle data, Oracle WebServer provides you a group of *PL/SQL* packages that you can use to easily generate Web pages from data stored in an Oracle database. These packages are called the PL / SQL Web Toolkit. The intent is for you to create PL / SQL procedures that access and process the Oracle data you wish to place on the Web. From within these procedures, you call the PL / SQL Web Toolkit procedures you need to create the HTML you want. You store the procedures you write in the database, just as other PL / SQL packages, including the toolkit, are stored. You also design your Web pages, including the dynamically-generated ones, to produce URLs that call the PL / SQL procedures you want in response to specified user actions. Having your code executed within the database brings many performance, security, and portability benefits. For more information on the PL / SQL Web Toolkit, see The PL / SQL Web Toolkit Reference.

CHAPTER

3 The Java Cartridge

Java is an object-oriented language for creating distributed applications on the Internet or other networks. Modules of Java code known as “applets” can be downloaded from the Internet or a local network in real time and locally executed. Java applets themselves can call and execute other applets, so that a Java application as executed on the user’s machine can be constructed “on the fly” from a repertoire of standard parts that reside on the net. You might call this the “building block” approach to programming. Among the important features of Java are the following:

- It is extremely portable. Java code is compiled to a form known as “bytecode”. This is a sort of generalized computer code that is not executable by any particular machine, but is recognized by the “Java Virtual Machine”. It is as bytecode that Java applets transverse the net. The Java Virtual Machine (VM) resides on the computer where the applet is to be executed and converts the bytecode to the native code for that machine. Currently, Java VMs exist or are planned for all current versions of Windows, Solaris, MacOS, OS/2, Linux, Amiga, and other platforms.
- It is fully object-oriented. Languages like C++ that add object-oriented features to non-object-oriented languages must make compromises. Java applets do not allow violation of object-oriented principles such as “encapsulation”.

- It uses a C-like syntax. This makes the language easier for C programmers to learn.
- It is multi-threaded, in effect executing several chains of control flow concurrently. The Java language itself provides tools for managing the threads, rather than relying exclusively on the OS.
- It prohibits direct memory manipulation. In Java, there are no pointers and no direct memory allocation. This eliminates a rich source of C's functionality, and an even richer source of its bugs.
- You can embed calls to Java applets in Web pages, and the applet will be executed by the browser, provided it is Java-enabled. Most major browsers plan to support Java.

Note: Though powerful, Java is a young technology. Oracle WebServer supports it because of its rich features and wide acceptance. However, you should be aware that it may be somewhat less stable than more mature technologies.

Client vs. Server Side Java

Oracle WebServer supports the use of Java either on the client, which is to say any Java-enabled Web browser, or on the server. Code to be executed on the client is for the most part extracted and manipulated like other data. The best way to handle such code is to store it in the OS file system and extract it in real time.

You can also execute Java as a WRB cartridge on the WebServer itself. You might want to do this, for example, to perform graphical manipulation for which PL/SQL is ill-suited. For example, you can combine several graphics from the database into a single image. Each region of the image would be a separate button that the user can click, and each button clicked would produce a different effect. In HTML, this is called an “image map”. Using Java on the server, you could generate such image maps dynamically, with the components of the image being based on the results of a database query.

To execute Java on the server, you use the WRB API to interface directly to the Java Interpreter residing in the WebServer. This interpreter finds and executes the Java code and returns the results, through the WRB interface, to the Web Listener.

To make it easier for you to develop Java applications, Oracle WebServer provides the Java Web Toolkit, a group of Java packages containing classes to aid in database access and dynamic HTML generation.

Using PL/SQL From Java

Since PL / SQL code is actually part of the database, you can call it from within Java, which enables you to create applications that combine the strengths of both languages. Because PL / SQL execution takes places in the database, doing this does not hinder the portability of the application. A PL / SQL application can execute without modification on any platform where the Oracle7 Server runs, just LiveHTML

LiveHTML is Oracle's implementation and extension of the standard Server Side Includes functionality defined by the NCSA. The LiveHTML Interpreter enables you to include dynamic content in otherwise static Web pages. At the point in your Web page where you want to interject dynamic content, you place a tag that points to one of the following:

- A static Web page.
- Another LiveHTML Web page.
- A script that is executed on the server and outputs HTML. This script may but need not conform to the *CGI* standard.
- A system variable, for example: FMODDATE.

You can use a variable to determine at runtime the Web page, variable, or script to which the tag points. This enables you to have a Web page that selects dynamically from among any number of static Web pages or scripts, based, for example, on values a user provides in an HTML form. The result is a dynamic Web page built of static Web page components, variables, and HTML output from scripts.

A Web page that is to use LiveHTML must be parsed by the WebServer. For this reason, it differs slightly from ordinary Web pages written in HTML, which the WebServer simply delivers to the browser. To have the LiveHTML tags executed on the server, you must use the WebServer Manager to specify that a given Web Listener is to parse files for LiveHTML. You have the option of having the Listener parse all files or just those with certain extensions.

Enabling users to execute scripts on the server can create security and other risks. For this reason, you can specify that a specific Listener allows only "crippled" includes. This means that LiveHTML parsed by that Listener will be able only to call static HTML, environment variables, or other server parsable files, not executable scripts.

You frequently use LiveHTML when you have standard components, such as menus, that you want on many pages. If desired, you can use LiveHTML to run the PL/SQL Agent under CGI, and thereby incorporate dynamic Oracle data in hardcoded Web pages.

CHAPTER

4 LiveHTML

LiveHTML is Oracle's implementation and extension of the standard Server Side Includes functionality defined by the NCSA. The LiveHTML Interpreter enables you to include dynamic content in otherwise static Web pages. At the point in your Web page where you want to interject dynamic content, you place a tag that points to one of the following:

- A static Web page.
- Another LiveHTML Web page.
- A script that is executed on the server and outputs HTML. This script may but need not conform to the *CGI* standard.
- A system variable, for example: FMODDATE.

You can use a variable to determine at runtime the Web page, variable, or script to which the tag points. This enables you to have a Web page that selects dynamically from among any number of static Web pages or scripts, based, for example, on values a user provides in an HTML form. The result is a dynamic Web page built of static Web page components, variables, and HTML output from scripts.

A Web page that is to use LiveHTML must be parsed by the WebServer. For this reason, it differs slightly from ordinary Web pages written in HTML, which the WebServer simply delivers to the browser. To have the LiveHTML tags executed on the server, you must use the WebServer Manager to specify that a given Web Listener is to parse files for LiveHTML. You have the option of having the Listener parse all files or just those with certain extensions.

Enabling users to execute scripts on the server can create security and other risks. For this reason, you can specify that a specific Listener allows only "crippled" includes. This means that LiveHTML parsed by that Listener will be able only to call static HTML, environment variables, or other server parsable files, not executable scripts.

You frequently use LiveHTML when you have standard components, such as menus, that you want on many pages. If desired, you can use LiveHTML to run the PL/SQL Agent under CGI, and thereby incorporate dynamic Oracle data in hardcoded Web pages.

CHAPTER

5

Web Request Broker Administration

The Oracle WebServer Manager includes several administration forms you can use to perform general Web Request Broker configuration, and to configure specific WRB cartridges. These forms allow you to:

- Configure a new WRB cartridge
- Modify a WRB cartridge configuration
- Delete a WRB cartridge

You can find the WRB administration WebServer Manager pages at:

http://*yourserver*:*port*/ows-abin/wrbadmin

where *yourserver*:*port* identifies the hostname and port on which your administration server is running. You need your administration username and password to access these and other WebServer Manager pages.

If you get a “not found” error when you try to access this URL, your WebServer might not be installed correctly. See [...installation manual] for more information.