IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	**PUBLIC VERSION**
Defendant/Counterclaim Plaintiff.	)	

APPENDIX TO EPICREALM'S MEMORANDUM IN OPPOSITION TO ORACLE'S MOTION FOR SUMMARY JUDGMENT OF NONINFRINGEMENT

VOLUME I OF II

EXHIBITS 1-17

Richard L. Horwitz (#2246)
David E. Moore (#3983)
POTTER ANDERSON & CORROON LLP
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Dated: August 21, 2008
Public Version Dated: August 28, 2008
880382 / 31393 / Oracle

TABLE OF CONTENTS

Exh.	*Document*
1.	U.S. Patent 5,894,554, Lowery et al.
2.	U.S. Patent 6,415,335, Lowery et al.
3.	Oracle's Proposed Claim Construction Chart, Dec. 14, 2007
4.	Order, Aug. 7, 2008, *EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.,* (E. D. Tex.)
5.	Excerpt from deposition transcript of Mark Clark, April 17, 2008
6.	Excerpt from deposition transcript of Paul C. Clark, July 11, 2008
7.	Excerpt from deposition transcript of David Finkel, June 20, 2008
8.	Excerpt from deposition transcript of Steve Harris, April 18, 2008
9.	Excerpt from deposition transcript of Michael Shamos, June 30, 2008
10.	Excerpt from deposition transcript of Douglas Surber, Feb. 28, 2008
11.	Excerpt from Expert Report Dr. Paul C. Clark, June 6, 2008
12.	Excerpt from Rebuttal Expert Report of David Finkel, June 6, 2006
13.	Excerpt from Expert Report Shamos
14.	Excerpt from Oracle9i Application Server Performance Guide Release 2 (9.0.2) April 2002, A95102-02, ORCL00847081-00847294
15.	Excerpt from Oracle Application Server Containers for J2EE 10g Release 2 (10.1.2), B14014-02, ORCL00444343-00444558
16.	Excerpt from Oracle Call Interface Programmer's Guide 11g Release 1 (11.1) B28395-03, May 2008
17.	Mod_oc4j Load Balancing in iAS 9.0.4, ORCL00018115-00018150
18.	Design Specification for OCI Runtime Connection Load Balancing in 11g, ORCL01310468-01310498
19.	Test Specification for Application Runtime and Recovery Performance using Runtime Load Balancing, Oracle 10gR2, ORCL01383645-01383660
20.	Oracle World, ORCL01050220-01050280
21.	Excerpt from Oracle Application Server 10g, Best Practices 10g (9.0.4) B12223-01, ORCL00384158-00384305

22. Oracle9iAS Web Cache Administrative and Deployment Guide, Release 2 (9.0.2) May 2002, A95404-02, ORCL00272342-00272887

23. Email from R. Vairavaraja, 8/24/07, ORCL01436981

24. Letter from J. Moore, FreshDirect, 3/11/08, EPIC436363

25. Excerpt from Oracle Application Server Best Practices Guide 10g Release 2 (10.1.2), B28654-01, March 2006, ORCL00528055-00528218

26. USSN 08/794,269, Office Action, 8/23/2000, ORCL01623264-01623276

27. USSN 09/615,383, Information Disclosure Statement, Dec. 12, 2005

28. Excerpt from Oracle Web Cache Administration and Deployment Guide, Release 1.0.2, A86722-01, Nov. 21, 2000, ORCL00107955-00108126

29. USSN 08/962,536, Information Disclosure Statement, Nov. 21, 2000 ORCL01622918-01622919

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 28, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on August 28, 2008, the attached document was Electronically Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Igor Shoiket
Townsend and Townsend and Crew LLP
Two Embarcadero Center
Eighth Floor
San Francisco, CA 94111-3834
OracleEpicrealm@townsend.com

Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Nitin Gupta
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Exhibit 1

US005894554A

United States Patent [19]

Lowery et al.

[11] Patent Number: 5,894,554

[45] Date of Patent: Apr. 13, 1999

[54] **SYSTEM FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS BY INTERCEPTING REQUEST AT WEB SERVER AND ROUTING TO PAGE SERVER THEREBY RELEASING WEB SERVER TO PROCESS OTHER REQUESTS**

[75] Inventors: **Keith Lowery**, Richardson; **Andrew B. Levine**, Plano; **Ronald L. Howell**, Rowlett, all of Tex.

[73] Assignee: **InfoSpinner, Inc.**, Richardson, Tex.

[21] Appl. No.: **08/636,477**

[22] Filed: **Apr. 23, 1996**

[51] **Int. Cl.**[6] **G06F 13/14**; G06F 13/20

[52] **U.S. Cl.** **395/200.33**; 395/200.68; 395/200.75; 395/680; 707/10; 707/104

[58] **Field of Search** 358/400; 395/800, 395/700, 200.68, 200.75, 200.53, 680, 200.33; 707/104, 10

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,866,706	9/1989	Christophersen et al.	370/85.7
5,341,499	8/1994	Doragh	395/700
5,392,400	2/1995	Berkowitz et al.	395/200
5,404,522	4/1995	Carmon et al.	395/650
5,404,523	4/1995	DellaFera et al.	395/650
5,404,527	4/1995	Irwin et al.	395/700
5,452,460	9/1995	Distelberg et al.	395/700
5,532,838	7/1996	Barbari	358/400
5,751,956	5/1998	Kirsch	395/200.33
5,761,673	6/1998	Bookman et al.	707/104

OTHER PUBLICATIONS

"Beyond the Web: Excavating the Real World Via Mosaic", Goldberg et al. Second International WWW, Oct. 17, 1994.

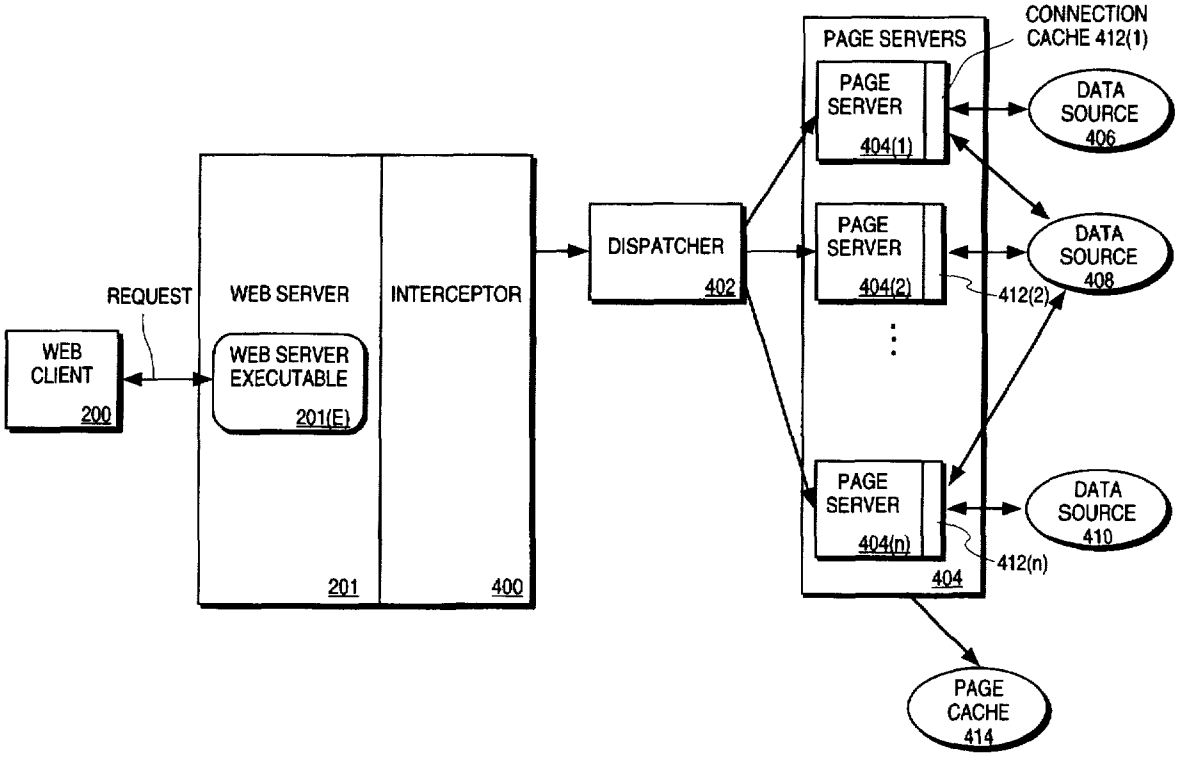
PCT International Search Report, Aug. 21, 1997.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Rehana Perveen
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman LLP

[57] **ABSTRACT**

The present invention teaches a method and apparatus for creating and managing custom Web sites. Specifically, one embodiment of the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources.

11 Claims, 5 Drawing Sheets



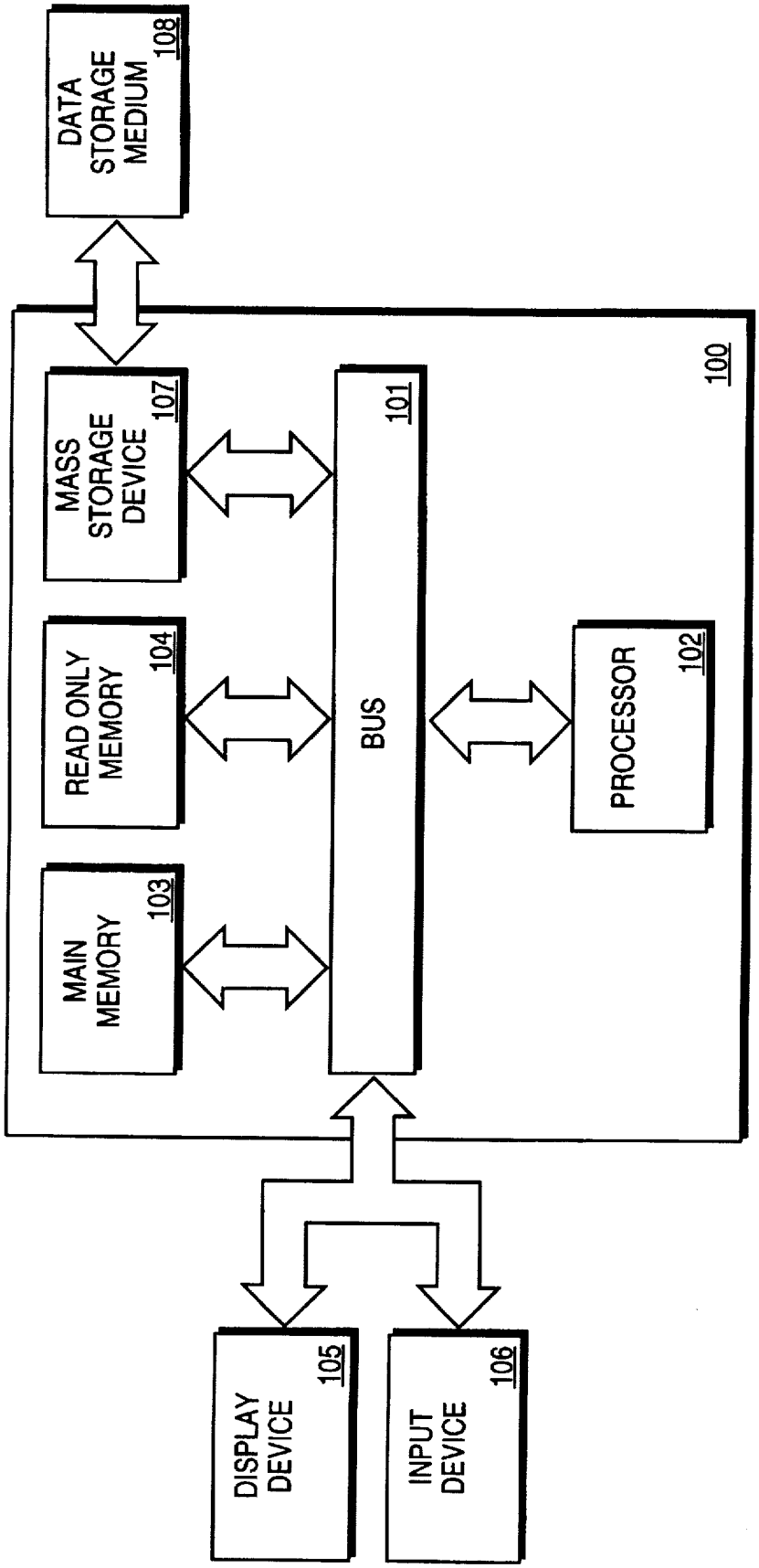

DATA STORAGE MEDIUM
108
100
MASS STORAGE DEVICE
107
BUS
101
READ ONLY MEMORY
104
PROCESSOR
102
MAIN MEMORY
103
DISPLAY DEVICE
105
INPUT DEVICE
106

FIG. 1

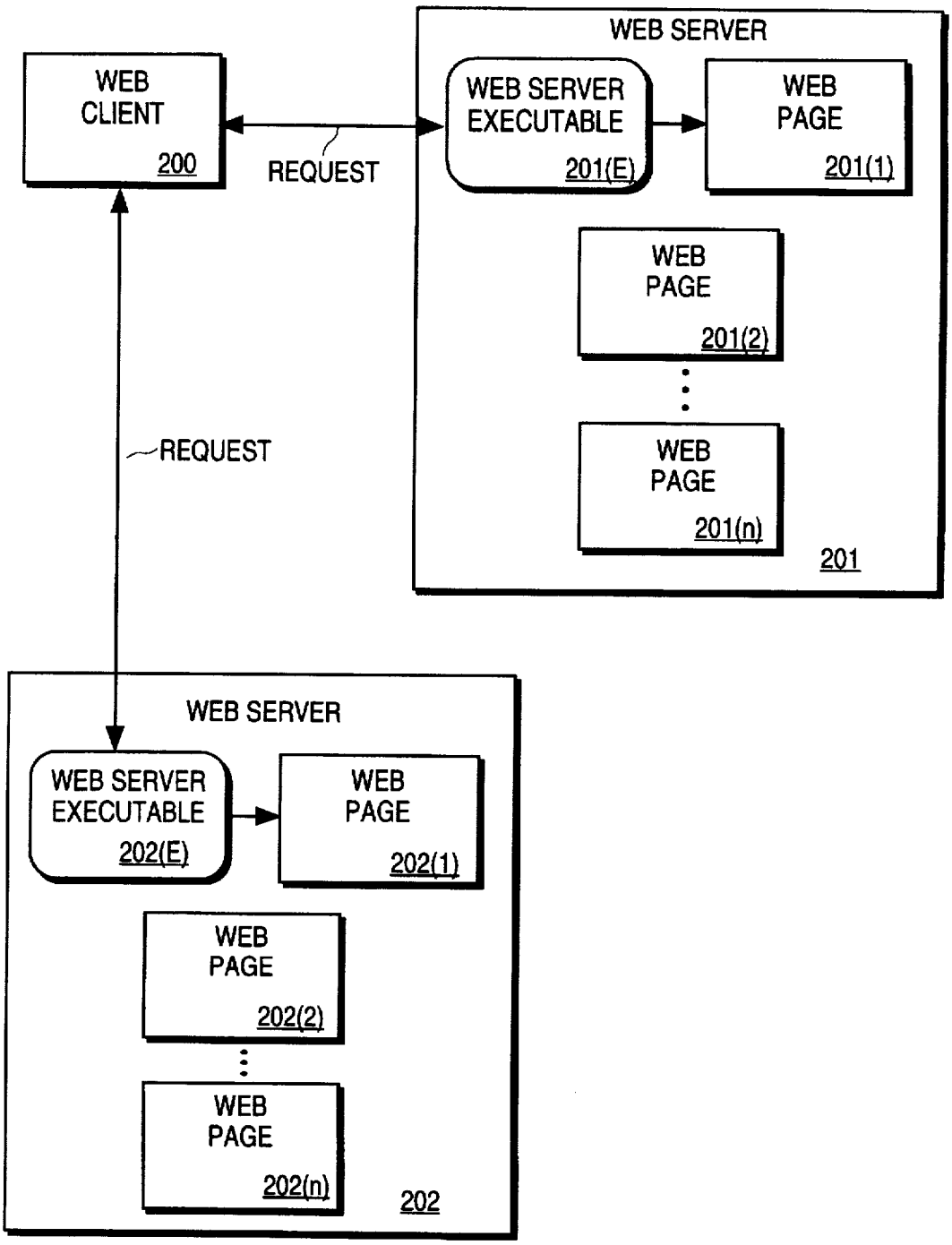


FIG. 2 (PRIOR ART)

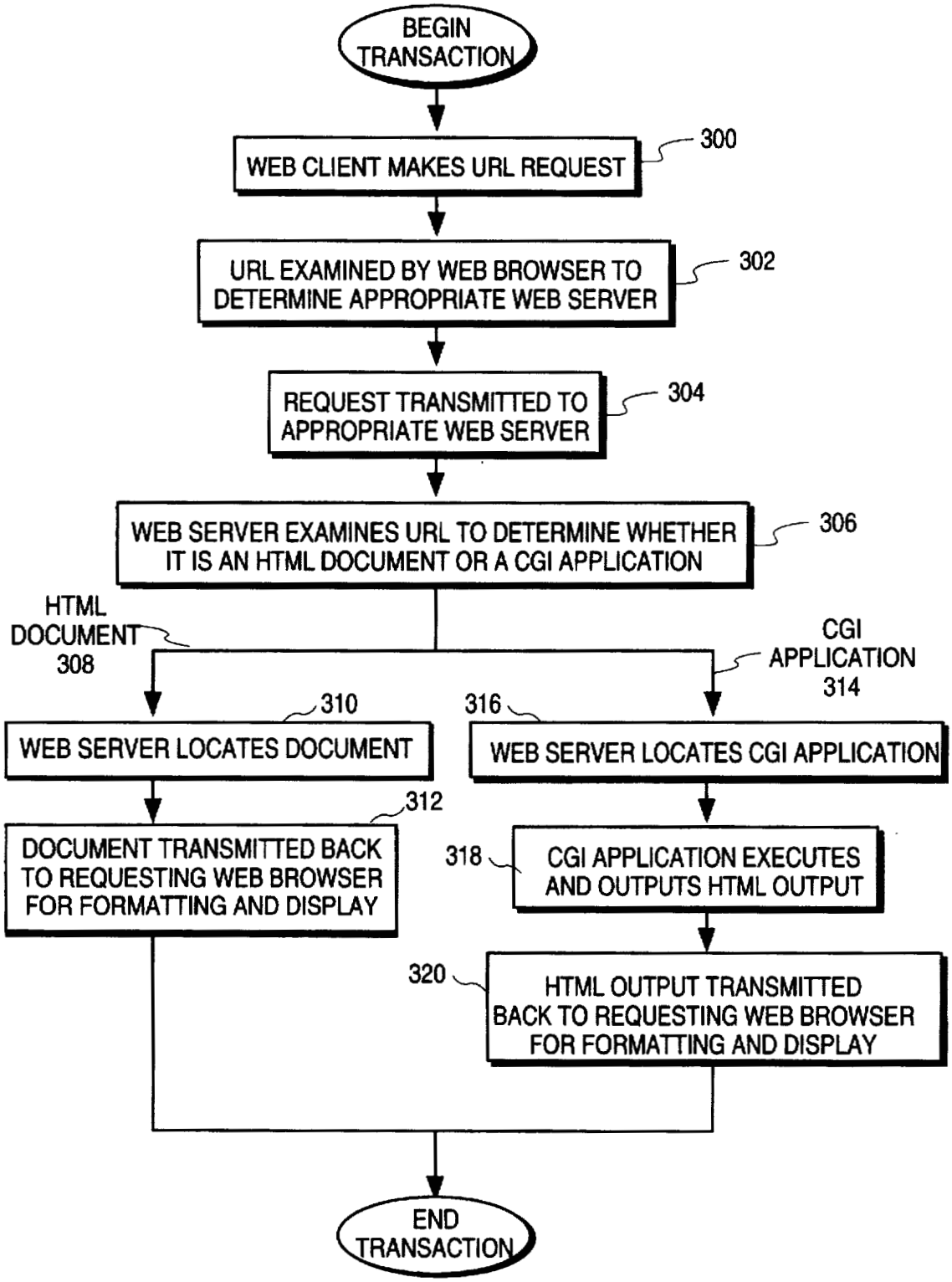


FIG. 3 (PRIOR ART)

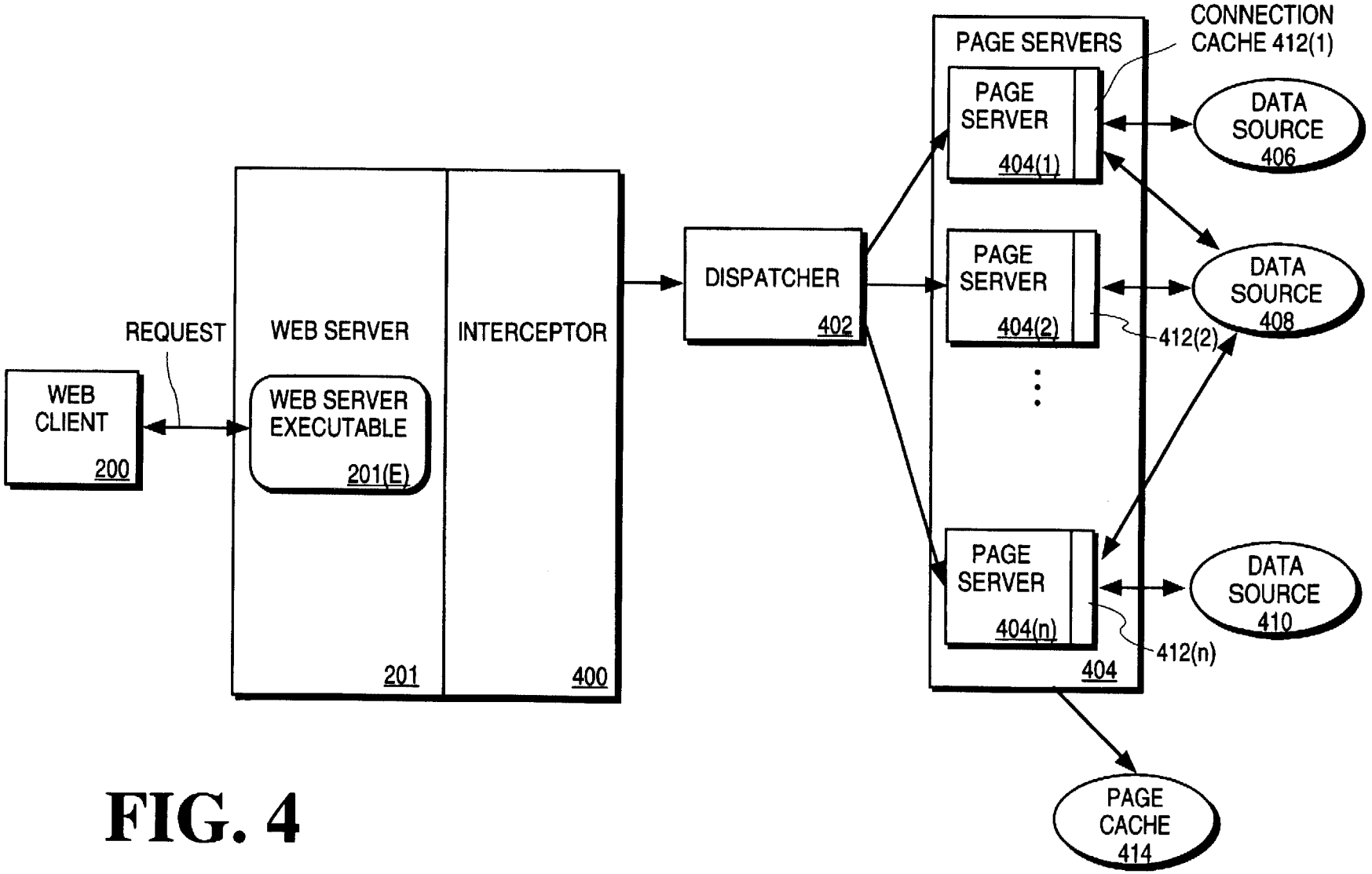

WEB CLIENT
200
REQUEST
WEB SERVER
WEB SERVER EXECUTABLE
201(E)
201
INTERCEPTOR
400
DISPATCHER
402
PAGE SERVERS
CONNECTION CACHE 412(1)
PAGE SERVER
404(1)
DATA SOURCE
406
PAGE SERVER
404(2)
412(2)
DATA SOURCE
408
PAGE SERVER
404(n)
412(n)
404
DATA SOURCE
410
PAGE CACHE
414

FIG. 4

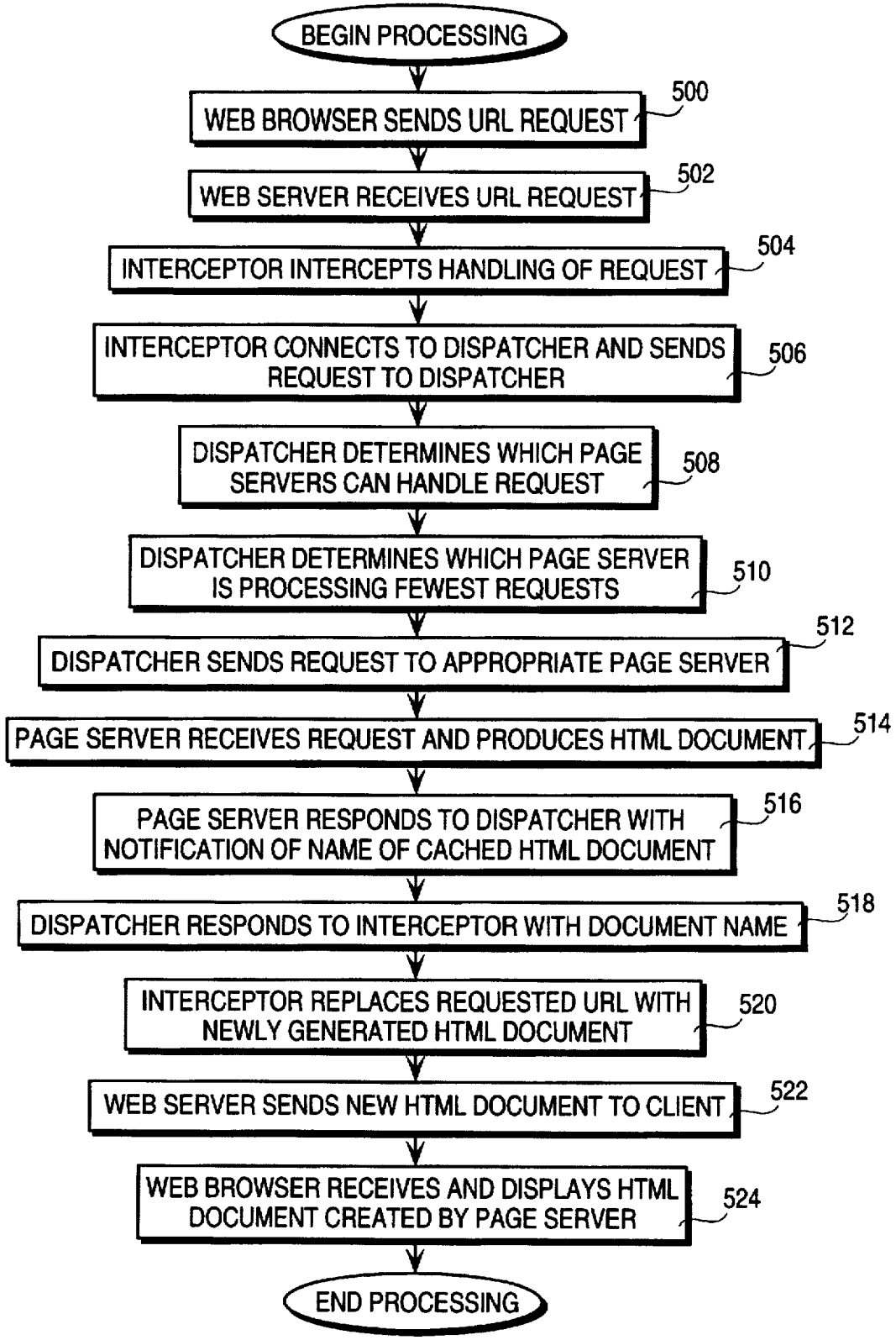

BEGIN PROCESSING
WEB BROWSER SENDS URL REQUEST
500
WEB SERVER RECEIVES URL REQUEST
502
INTERCEPTOR INTERCEPTS HANDLING OF REQUEST
504
INTERCEPTOR CONNECTS TO DISPATCHER AND SENDS REQUEST TO DISPATCHER
506
DISPATCHER DETERMINES WHICH PAGE SERVERS CAN HANDLE REQUEST
508
DISPATCHER DETERMINES WHICH PAGE SERVER IS PROCESSING FEWEST REQUESTS
510
DISPATCHER SENDS REQUEST TO APPROPRIATE PAGE SERVER
512
PAGE SERVER RECEIVES REQUEST AND PRODUCES HTML DOCUMENT
514
PAGE SERVER RESPONDS TO DISPATCHER WITH NOTIFICATION OF NAME OF CACHED HTML DOCUMENT
516
DISPATCHER RESPONDS TO INTERCEPTOR WITH DOCUMENT NAME
518
INTERCEPTOR REPLACES REQUESTED URL WITH NEWLY GENERATED HTML DOCUMENT
520
WEB SERVER SENDS NEW HTML DOCUMENT TO CLIENT
522
WEB BROWSER RECEIVES AND DISPLAYS HTML DOCUMENT CREATED BY PAGE SERVER
524
END PROCESSING

FIG. 5

SYSTEM FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS BY INTERCEPTING REQUEST AT WEB SERVER AND ROUTING TO PAGE SERVER THEREBY RELEASING WEB SERVER TO PROCESS OTHER REQUESTS

FIELD OF THE INVENTION

The present invention relates to the field of Internet technology. Specifically, the present invention relates to the creation and management of custom World Wide Web sites.

DESCRIPTION OF RELATED ART

The World Wide Web (the Web) represents all of the computers on the Internet that offer users access to information on the Internet via interactive documents or Web pages. These Web pages contain hypertext links that are used to connect any combination of graphics, audio, video and text, in a non-linear, non-sequential manner. Hypertext links are created using a special software language known as HyperText Mark-Up Language (HTML).

Once created, Web pages reside on the Web, on Web servers or Web sites. A Web site can contain numerous Web pages. Web client machines running Web browsers can access these Web pages at Web sites via a communications protocol known as HyperText Transport Protocol (HTTP). Web browsers are software interfaces that run on World Wide Web clients to allow access to Web sites via a simple user interface. A Web browser allows a Web client to request a particular Web page from a Web site by specifying a Uniform Resource Locator (URL). A URL is a Web address that identifies the Web page and its location on the Web. When the appropriate Web site receives the URL, the Web page corresponding to the requested URL is located, and if required, HTML output is generated. The HTML output is then sent via HTTP to the client for formatting on the client's screen.

Although Web pages and Web sites are extremely simple to create, the proliferation of Web sites on the Internet highlighted a number of problems. The scope and ability of a Web page designer to change the content of the Web page was limited by the static nature of Web pages. Once created, a Web page remained static until it was manually modified. This in turn limited the ability of Web site managers to effectively manage their Web sites.

The Common Gateway Interface (CGI) standard was developed to resolve the problem of allowing dynamic content to be included in Web pages. CGI "calls" or procedures enable applications to generate dynamically created HTML output, thus creating Web pages with dynamic content. Once created, these CGI applications do not have to be modified in order to retrieve "new" or dynamic data. Instead, when the Web page is invoked, CGI "calls" or procedures are used to dynamically retrieve the necessary data and to generate a Web page.

CGI applications also enhanced the ability of Web site administrators to manage Web sites. Administrators no longer have to constantly update static Web pages. A number of vendors have developed tools for CGI based development, to address the issue of dynamic Web page generation. Companies like Spider™ and Bluestone™, for example, have each created development tools for CGI-based Web page development. Another company, Haht Software™, has developed a Web page generation tool that uses a BASIC-like scripting language, instead of a CGI scripting language.

Tools that generate CGI applications do not, however, resolve the problem of managing numerous Web pages and requests at a Web site. For example, a single company may maintain hundreds of Web pages at their Web site. Current Web server architecture also does not allow the Web server to efficiently manage the Web page and process Web client requests. Managing these hundreds of Web pages in a coherent manner and processing all requests for access to the Web pages is thus a difficult task. Existing development tools are limited in their capabilities to facilitate dynamic Web page generation, and do not address the issue of managing Web requests or Web sites.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a method and apparatus for creating and managing custom Web sites. Specifically, the present invention claims a method and apparatus for managing dynamic web page generation requests.

In one embodiment, the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources. Other embodiments also include connection caches to the one or more data sources, page caches for each page server, and custom HTML extension templates for configuring the Web page.

Other objects, features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical computer system in which the present invention operates.

FIG. **2** illustrates a typical prior art Web server environment.

FIG. **3** illustrates a typical prior art Web server environment in the form of a flow diagram.

FIG. **4** illustrates one embodiment of the presently claimed invention.

FIG. **5** illustrates the processing of a Web browser request in the farm of a flow diagram, according to one embodiment of the presently claimed invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention relates to a method and apparatus for creating and managing custom Web sites. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent to one of ordinary skill in the art, however, that these specific details need not be used to practice the present invention. In other instances, well-known structures, interfaces and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

FIG. **1** illustrates a typical computer system **100** in which the present invention operates. The preferred embodiment of

the present invention is implemented on an IBM™ Personal Computer manufactured by IBM Corporation of Armonk, N.Y. An alternate embodiment may be implemented on an RS/6000™ Workstation manufactured by IBM Corporation of Armonk, N.Y. It will be apparent to those of ordinary skill in the art that other computer system architectures may also be employed.

In general, such computer systems as illustrated by FIG. 1 comprise a bus **101** for communicating information, a processor **102** coupled with the bus **101** for processing information, main memory **103** coupled with the bus **101** for storing information and instructions for the processor **102**, a read-only memory **104** coupled with the bus **101** for storing static information and instructions for the processor **102**, a display device **105** coupled with the bus **101** for displaying information for a computer user, an input device **106** coupled with the bus **101** for communicating information and command selections to the processor **102**, and a mass storage device **107**, such as a magnetic disk and associated disk drive, coupled with the bus **101** for storing information and instructions. A data storage medium **108** containing digital information is configured to operate with mass storage device **107** to allow processor **102** access to the digital information on data storage medium **108** via bus **101**.

Processor **102** may be any of a wide variety of general purpose processors or microprocessors such as the Pentium™ microprocessor manufactured by Intel™ Corporation or the RS/6000™ processor manufactured by IBM Corporation. It will be apparent to those of ordinary skill in the art, however, that other varieties of processors may also be used in a particular computer system. Display device **105** may be a liquid crystal device, cathode ray tube (CRT), or other suitable display device. Mass storage device **107** may be a conventional hard disk drive, floppy disk drive, CD-ROM drive, or other magnetic or optical data storage device for reading and writing information stored on a hard disk, a floppy disk, a CD-ROM a magnetic tape, or other magnetic or optical data storage medium. Data storage medium **108** may be a hard disk, a floppy disk, a CD-ROM, a magnetic tape, or other magnetic or optical data storage medium.

In general, processor **102** retrieves processing instructions and data from a data storage medium **108** using mass storage device **107** and downloads this information into random access memory **103** for execution. Processor **102**, then executes an instruction stream from random access memory **103** or read-only memory **104**. Command selections and information input at input device **106** are used to direct the flow of instructions executed by processor **102**. Equivalent input device **106** may also be a pointing device such as a conventional mouse or trackball device. The results of this processing execution are then displayed on display device **105**.

The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system **100** in a conventional manner. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium **108** and subsequently loaded into and executed within computer system **100**. Once initiated, the software of the preferred embodiment operates in the manner described below.

FIG. **2** illustrates a typical prior art Web server environment. Web client **200** can make URL requests to Web server **201** or Web server **202**. Web servers **201** and **202** include Web server executables, **201**(E) and **202**(E) respectively, that perform the processing of Web client requests. Each Web server may have a number of Web pages **201**(1)–(n) and **202**(1)–(n). Depending on the URL specified by the Web client **200**, the request may be routed by either Web server executable **201**(E) to Web page **201** (1), for example, or from Web server executable **202**(E) to Web page **202** (1). Web client **200** can continue making URL requests to retrieve other Web pages. Web client **200** can also use hyperlinks within each Web page to "jump" to other Web pages or to other locations within the same Web page.

FIG. **3** illustrates this prior art Web server environment in the form of a flow diagram. In processing block **300**, the Web client makes a URL request. This URL request is examined by the Web browser to determine the appropriate Web server to route the request to in processing block **302**. In processing block **304** the request is then transmitted from the Web browser to the appropriate Web server, and in processing block **306** the Web server executable examines the URL to determine whether it is a HTML document or a CGI application. If the request is for an HTML document **308**, then the Web server executable locates the document in processing block **310**. The document is then transmitted back through the requesting Web browser for formatting and display in processing block **312**.

If the URL request is for a CGI application **314**, however, the Web server executable locates the CGI application in processing block **316**. The CGI application then executes and outputs HTML output in processing block **318** and finally, the HTML output is transmitted back to requesting Web browser for formatting and display in processing block **320**.

This prior art Web server environment does not, however, provide any mechanism for managing the Web requests or the Web sites. As Web sites grow, and as the number of Web clients and requests increase, Web site management becomes a crucial need.

For example, a large Web site may receive thousands of requests or "hits" in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running "multi-threaded" operating systems that allow transactions to be processed by independent "threads," all the threads are nevertheless on a single machine, sharing a processor. As such, the Web executable thread may hand off a request to a processing thread, but both threads will still have to be handled by the processor on the Web server machine. When numerous requests are being simultaneously processed by multiple threads on a single machine, the Web server can slow down significantly and become highly inefficient. The claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers.

FIG. **4** illustrates one embodiment of the presently claimed invention. Web client **200** issues a URL request that is processed to determined proper routing. In this embodiment, the request is routed to Web server **201**. Instead of Web server executable **201**(E) processing the URL request, however, Interceptor **400** intercepts the request and routes it to Dispatcher **402**. In one embodiment, Interceptor **400** resides on the Web server machine as an extension to Web server **201**. This embodiment is appropriate for Web servers such as Netsite™ from Netscape, that support such extensions. A number of public domain Web servers, such as NCSA™ from the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign, however, do not provide support for

this type of extension. Thus, in an alternate embodiment, Interceptor **400** is an independent module, connected via an "intermediate program" to Web server **201**. This intermediate program can be a simple CGI application program that connects Interceptor **400** to Web server **201**. Alternate intermediate programs the perform the same functionality can also be implemented.

In one embodiment of the invention, Dispatcher **402** resides on a different machine than Web server **201**. This embodiment overcomes the limitation described above, in prior art Web servers, wherein all processing is performed by the processor on a single machine. By routing the request to Dispatcher **402** residing on a different machine than the Web server executable **201**(E), the request can then be processed by a different processor than the Web server executable **201**(E). Web server executable **201**(E) is thus free to continue servicing client requests on Web server **201** while the request is processed "off-line," at the machine on which Dispatcher **402** resides.

Dispatcher **402** can, however, also reside on the same machine as the Web server. The Web site administrator has the option of configuring Dispatcher **402** on the same machine as Web server **201**, taking into account a variety of factors pertinent to a particular Web site, such as the size of the Web site, the number of Web pages and the number of hits at the Web site. Although this embodiment will not enjoy the advantage described above, namely off-loading the processing of Web requests from the Web server machine, the embodiment does allow flexibility for a small Web site to grow. For example, a small Web site administrator can use a single machine for both Dispatcher **402** and Web server **201** initially, then off-load Dispatcher **402** onto a separate machine as the Web site grows. The Web site can thus take advantage of other features of the present invention regardless of whether the site has separate machines configured as Web servers and dispatchers.

Dispatcher **402** receives the intercepted request and then dispatches the request to one of a number of Page servers **404** (1)–(n). For example, if Page server **404** (1) receives the dispatched request, it processes the request and retrieves the data from an appropriate data source, such as data source **406**, data source **408**, or data source **410**. Data sources, as used in the present application, include databases, spreadsheets, files and any other type of data repository. Page server **404** (1) can retrieve data from more than one data source and incorporate the data from these multiple data sources in a single Web page.

In one embodiment, each Page server **404**(1)–(n) resides on a separate machine on the network to distribute the processing of the request. Dispatcher **402** maintains a variety of information regarding each Page server on the network, and dispatches requests based on this information. For example, Dispatcher **402** retains dynamic information regarding the data sources that any given Page server can access. Dispatcher **402** thus examines a particular request and determines which Page servers can service the URL request. Dispatcher **402** then hands off the request to the appropriate Page server.

For example, if the URL request requires financial data from data source **408**, dispatcher **402** will first examine an information list. Dispatcher **402** may determine that Page server **404**(3), for example, has access to the requisite data in data source **408**. Dispatcher **402** will thus route the URL request to Page server **404**(3). This "connection caching" functionality is described in more detail below, under the heading "Performance."

Alternately, Dispatcher **402** also has the ability to determine whether a particular Page server already has the necessary data cached in the Page server's page cache (described in more detail below, under the heading "Performance"). Dispatcher **402** may thus determine that Page server **404**(1) and **404**(2) are both logged into Data source **408**, but that Page server **404**(2) has the financial information already cached in Page server **404**(2)'s page cache. In this case, Dispatcher **402** will route the URL request to Page server **404**(2) to more efficiently process the request.

Finally, Dispatcher **402** may determine that a number or all Page servers **404**(1)–(n) are logged into Data source **408**. In this scenario, Dispatcher **402** can examine the number of requests that each Page server is servicing and route the request to the least busy page server. This "load balancing" capability can significantly increase performance at a busy Web site and is discussed in more detail below, under the heading "Scalability".

If, for example, Page server **404**(2), receives the request, Page server **404**(2) will process the request. While Page server **404**(2) is processing the request, Web server executable **201**(E) can concurrently process other Web client requests. This partitioned architecture thus allows both Page server **404**(2) and Web server executable **201**(E) to simultaneously process different requests, thus increasing the efficiency of the Web site. Page server **404**(2) dynamically generates a Web page in response to the Web client request, and the dynamic Web page is then either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

One embodiment of the claimed invention also provides a Web page designer with HTML extensions, or "dyna" tags. These dyna tags provide customized HTML functionality to a Web page designer, to allow the designer to build customized HTML templates that specify the source and placement of retrieved data. For example, in one embodiment, a "dynatext" HTML extension tag specifies a data source and a column name to allow the HTML template to identify the data source to log into and the column name from which to retrieve data. Alternatively, "dyna-anchor" tags allow the designer to build hyperlink queries while "dynablock" tags provide the designer with the ability to iterate through blocks of data. Page servers use these HTML templates to create dynamic Web pages. Then, as described above, these dynamic Web pages are either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

The presently claimed invention provides numerous advantages over prior art Web servers, including advantages in the areas of performance, security, extensibility and scalability.

Performance

One embodiment of the claimed invention utilizes connection caching and page caching to improve performance. Each Page server can be configured to maintain a cache of connections to numerous data sources. For example, as illustrated in FIG. **4**, Page server **404**(1) can retrieve data from data source **406**, data source **408** or data source **410**. Page server **404**(1) can maintain connection cache **412**(1), containing connections to each of data source **406**, data source **408** and data source **410**, thus eliminating connect times from the Page servers to those data sources.

Additionally, another embodiment of the present invention supports the caching of finished Web pages, to optimize

the performance of the data source being utilized. This "page caching" feature, illustrated in FIG. **4** as Page cache **414**, allows the Web site administrator to optimize the performance of data sources by caching Web pages that are repeatedly accessed. Once the Web page is cached, subsequent requests or "hits" will utilize the cached Web page rather than re-accessing the data source. This can radically improve the performance of the data source.

Security

The present invention allows the Web site administrator to utilize multiple levels of security to manage the Web site. In one embodiment, the Page server can utilize all standard encryption and site security features provided by the Web server. In another embodiment, the Page server can be configured to bypass connection caches **412**(1)–(n), described above, for a particular data source and to require entry of a user-supplied identification and password for the particular data source the user is trying to access.

Additionally, another embodiment of the presently claimed invention requires no real-time access of data sources. The Web page caching ability, described above, enables additional security for those sites that want to publish non-interactive content from internal information systems, but do not want real-time Internet accessibility to those internal information systems. In this instance, the Page server can act as a "replication and staging gent" and create Web pages in batches, rather than in real-time. These "replicated" Web pages are then "staged" for access at a later time, and access o the Web pages in this scenario is possible even if the Page server and dispatcher are not present later.

In yet another embodiment, the Page server can make a single pass through a Web library, and compile a Web site that exists in the traditional form of separately available files. A Web library is a collection of related Web books and Web pages. More specifically, the Web library is a hierarchical organization of Web document templates, together with all the associated data source information. Information about an entire Web site is thus contained in a single physical file, thus simplifying the problem of deploying Web sites across multiple Page servers. The process of deploying the Web site in this embodiment is essentially a simple copy of a single file.

Extensibility

One embodiment of the present invention provides the Web site administrator with Object Linking and Embedding (OLE) 2.0 extensions to extend the page creation process. These OLE 2.0 extensions also allow information submitted over the Web to be processed with user-supplied functionality. Utilizing development tools such as Visual Basic, Visual C++ or PowerBuilder that support the creation of OLE 2.0 automation, the Web site administrator can add features and modify the behavior of the Page servers described above. This extensibility allows one embodiment of the claimed invention to be incorporated with existing technology to develop an infinite number of custom web servers.

For example, OLE 2.0 extensions allow a Web site administrator to encapsulate existing business rules in an OLE 2.0 automation interface, to be accessed over the Web. One example of a business rule is the steps involved in the payoff on an installment or mortgage loan. The payoff may involve, for example, taking into account the current balance, the date and the interest accrued since the last payment. Most organizations already have this type of business rule implemented using various applications, such as Visual Basic for client-server environments, or CICS programs on mainframes. If these applications are OLE 2.0 compliant, the Page server "dynaobject" HTML extension tag can be used to encapsulated the application in an OLE 2.0 automation interface. The Page server is thus extensible, and can incorporate the existing application with the new Page server functionality.

Scalability

One embodiment of the claimed invention allows "plug and play" scalability. As described above, referring to FIG. **4**, Dispatcher **402** maintains information about all the Page servers configured to be serviced by Dispatcher **402**. Any number of Page servers can thus be "plugged" into the configuration illustrated in FIG. **4**, and the Page servers will be instantly activated as the information is dynamically updated in Dispatcher **402**. The Web site administrator can thus manage the overhead of each Page server and modify each Page server's load, as necessary, to improve performance. In this manner, each Page server will cooperate with other Page servers within a multi-server environment. Dispatcher **402** can examine the load on each Page server and route new requests according to each Page server's available resources. This "load-balancing" across multiple Page servers can significantly increase a Web site's performance.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention. A Web browser sends a URL request to a Web server in processing block **500**. In processing block **502**, the Web server receives the URL request, and an interceptor then intercepts the handling of the request in processing block **504**. The interceptor connects to a dispatcher and sends the URL request to the dispatcher in processing block **506**. In processing block **508**, the dispatcher determines which Page servers can handle the request. The dispatcher also determines which Page server is processing the fewest requests in processing block **510**, and in processing block **512**, the dispatcher sends the URL request to an appropriate Page server. The Page server receives the request and produces an HTML document in processing block **514**. The Page server then responds to the dispatcher with notification of the name of the cached HTML document in processing block **516**. In processing block **518**, the dispatcher responds to the interceptor with the document name, and the interceptor then replaces the requested URL with the newly generated HTML document in processing block **520**. The Web server then sends the new HTML document to the requesting client in processing block **522**. Finally, the Web browser receives and displays the HTML document created by the Page server at processing block **524**.

Thus, a method and apparatus for creating and managing custom Web sites is disclosed. These specific arrangements and methods described herein are merely illustrative of the principles of the present invention. Numerous modifications in form and detail may be made by those of ordinary skill in the art without departing from the scope of the present invention. Although this invention has been shown in relation to a particular preferred embodiment, it should not be considered so limited. Rather, the present invention is limited only by the scope of the appended claims.

We claim:

1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

routing said request from said Web server to a page server, said page server receiving said request and releasing

said Web server to process other requests, wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

2. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

3. The computer-implemented method in claim **2** wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

4. The computer-implemented method in claim **3** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

5. The computer-implemented method in claim **3** wherein said step of processing said request includes the step of logging into said one or more data sources.

6. The computer-implemented method in claim **3** wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

7. The computer-implemented method in claim **3** wherein said page server includes custom HTML extension templates for configuring said Web page.

8. The computer-implemented method in claim **7** wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

9. A networked system for managing a dynamic Web page generation request, said system comprising:

one or more data sources;

a page server having a processing means;

a first computer system including means for generating said request; and

a second computer system including means for receiving said request from said first computer, said second computer system also including a router, said router routing said request from said second computer system to said page server, wherein said routing further includes intercepting said request at said second computer, routing said request from said second computer to a dispatcher, and dispatching said request to said page server said page server receiving said request and releasing said second computer system to process other requests, said page server processing means processing said request and dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from said one or more data sources.

10. The networked system in claim **9** wherein said router in said second computer system includes:

an interceptor intercepting said request at said second computer system and routing said request; and

a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

11. A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

routing a dynamic Web page generation request from a Web server to a page server, said page server receiving said request and releasing said Web server to process other requests wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

* * * * *

Exhibit 2

US006415335B1

(12) **United States Patent**
Lowery et al.

(10) **Patent No.: US 6,415,335 B1**
(45) **Date of Patent: *Jul. 2, 2002**

(54) **SYSTEM AND METHOD FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS**

(75) Inventors: **Keith Lowery**, Richardson; **Andrew B. Levine**, Plano; **Ronald L. Howell**, Rowlett, all of TX (US)

(73) Assignee: **epicRealm Operating Inc.**, Richardson, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/234,048**

(22) Filed: **Jan. 19, 1999**

Related U.S. Application Data

(62) Division of application No. 08/636,477, filed on Apr. 23, 1996, now Pat. No. 5,894,554.

(51) **Int. Cl.**[7] **G06F 13/14**; G06F 13/20
(52) **U.S. Cl.** **710/5**; 710/7; 709/219; 709/223; 709/238
(58) **Field of Search** 709/238, 223, 709/219; 710/5, 7, 20–21

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,866,706	A		9/1989	Christophersen et al. .. 370/85.7
5,341,499	A		8/1994	Doragh 395/700
5,392,400	A		2/1995	Berkowitz et al. 395/200
5,404,522	A		4/1995	Carmon et al. 395/650
5,404,523	A		4/1995	DellaFera et al. 395/650
5,404,527	A	*	4/1995	Irwin et al. 395/700
5,452,460	A		9/1995	Distelberg et al. 395/700
5,532,838	A		7/1996	Barbari 358/400
5,701,463	A	*	12/1997	Malcolm 395/610
5,751,956	A		5/1998	Kirsch 395/200.33
5,752,246	A	*	5/1998	Rogers et al. 707/10
5,754,772	A	*	5/1998	Leaf 395/200.33
5,761,673	A	*	6/1998	Bookman et al. 707/104
5,774,660	A		6/1998	Brendel et al. 395/200.31
5,774,668	A		6/1998	Choquier et al. 395/200.53

OTHER PUBLICATIONS

Hoffner, 'Inter–operability and distributed application platform design', Web URL:http://www.ansa.co.uk/, 1995, pp. 342–356.*

Mourad et al, 'Scalable Web Server Architectures', IEEE, Jun. 1997, pp. 12–16.*

Dias et al, 'A Scalable and Highly Available Web Server', IEEE, 1996, pp. 85–92.*

'Single System Image and Load Balancing for Network Access to a Loosely Coupled Complex', IBM TDB, vol. 34, Feb. 1992, pp. 464–467.*

Dias, Daniel M., et al.; A Scalable and Highly Available Web Server; IBM Research Division; T.J. Watson Research Center; 7 pages.

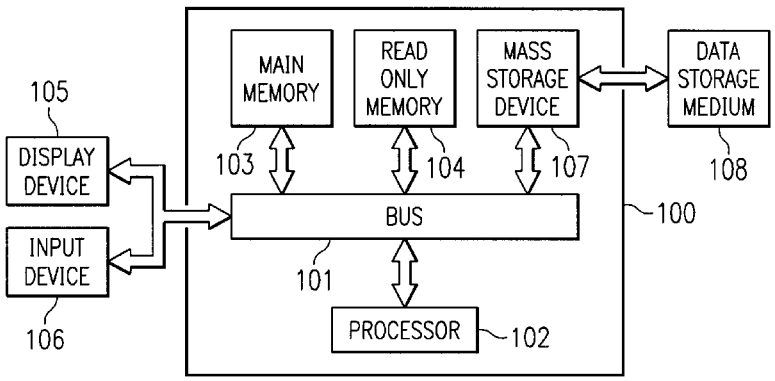
(List continued on next page.)

Primary Examiner—Jeffrey Gaffin
Assistant Examiner—Rehana Perveen
(74) *Attorney, Agent, or Firm*—Baker Botts L.L.P.

(57) **ABSTRACT**

The present invention teaches a method and apparatus for creating and managing custom Web sites. Specifically, one embodiment of the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources.

29 Claims, 4 Drawing Sheets



OTHER PUBLICATIONS

Andresen, Daniel, Et Al.; Scalability Issues for High Performance Digital Libraries on the World Wide Web; Department of Computer Science; University of California at Santa Barbara; 10 pages.

Andresen, Daniel, Et Al.; SWEB: Towards a Scalable World Wide Web Server on Multicomputers; Department of Computer Science; University of California at Santa Barbara; 7 pages.

Holmedahl, Vegard; Et Al.; Cooperative Caching of Dynamic Content on a Distributed Web Server; Department of Computer Science; University of California at Santa Barbara; 8 pages.

Overson, Nicole; NeXT Ships WebObjects—On Time—As Promised; Deja.com: NeXT Ships WebObjects—On Time—As Promishttp://X28..deja.com/=dnc/ST_m=ps...EXT=927585438. 1744765032&hitnum=33.

International Search Report; 7 pages; dated Aug. 21, 1997.

Birman, Kenneth P. and van Renesse, Robbert; Software for Reliable Networks; Scientific American; May 1996; pp. 64–69.

"Beyond the Web: Excavating the Real World Via Mosaic"; Goldberg et al.; Second International WWW Conference; Oct. 17, 1994.

* cited by examiner

FIG. 1

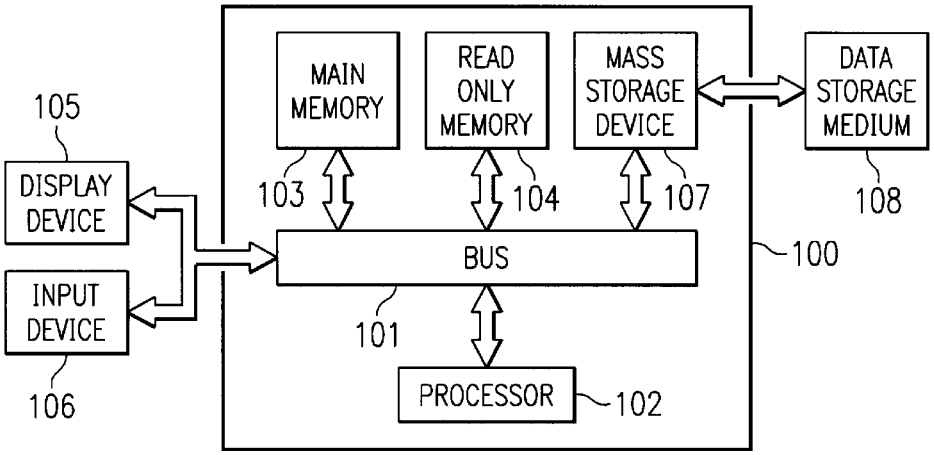


FIG. 2
(PRIOR ART)

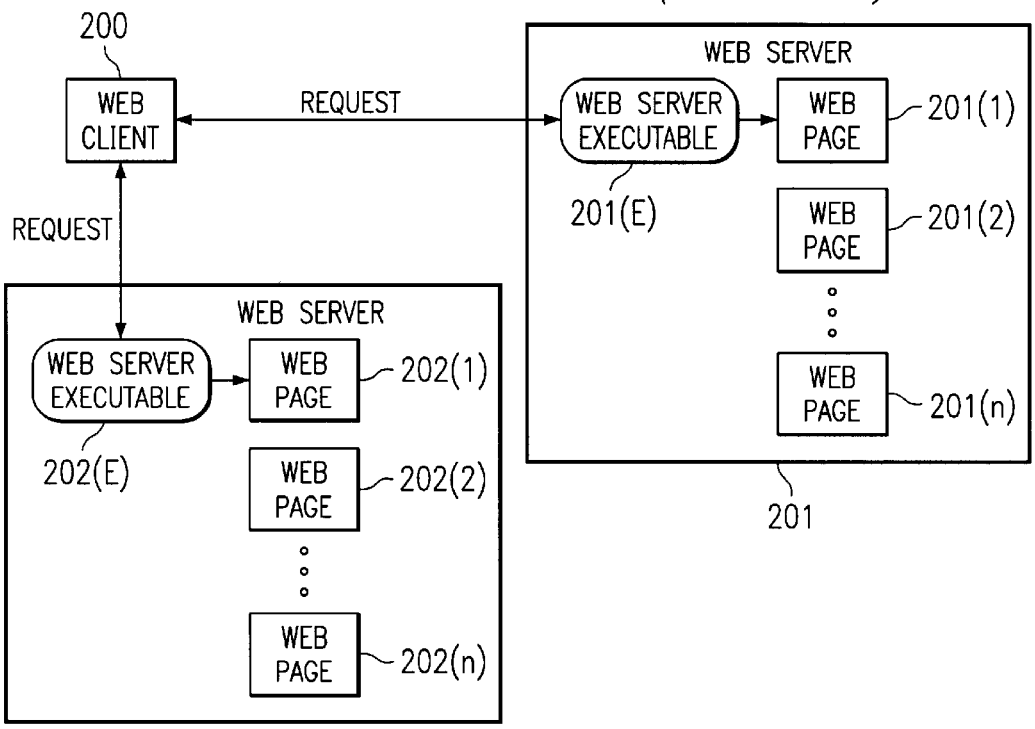


FIG. 3
(PRIOR ART)

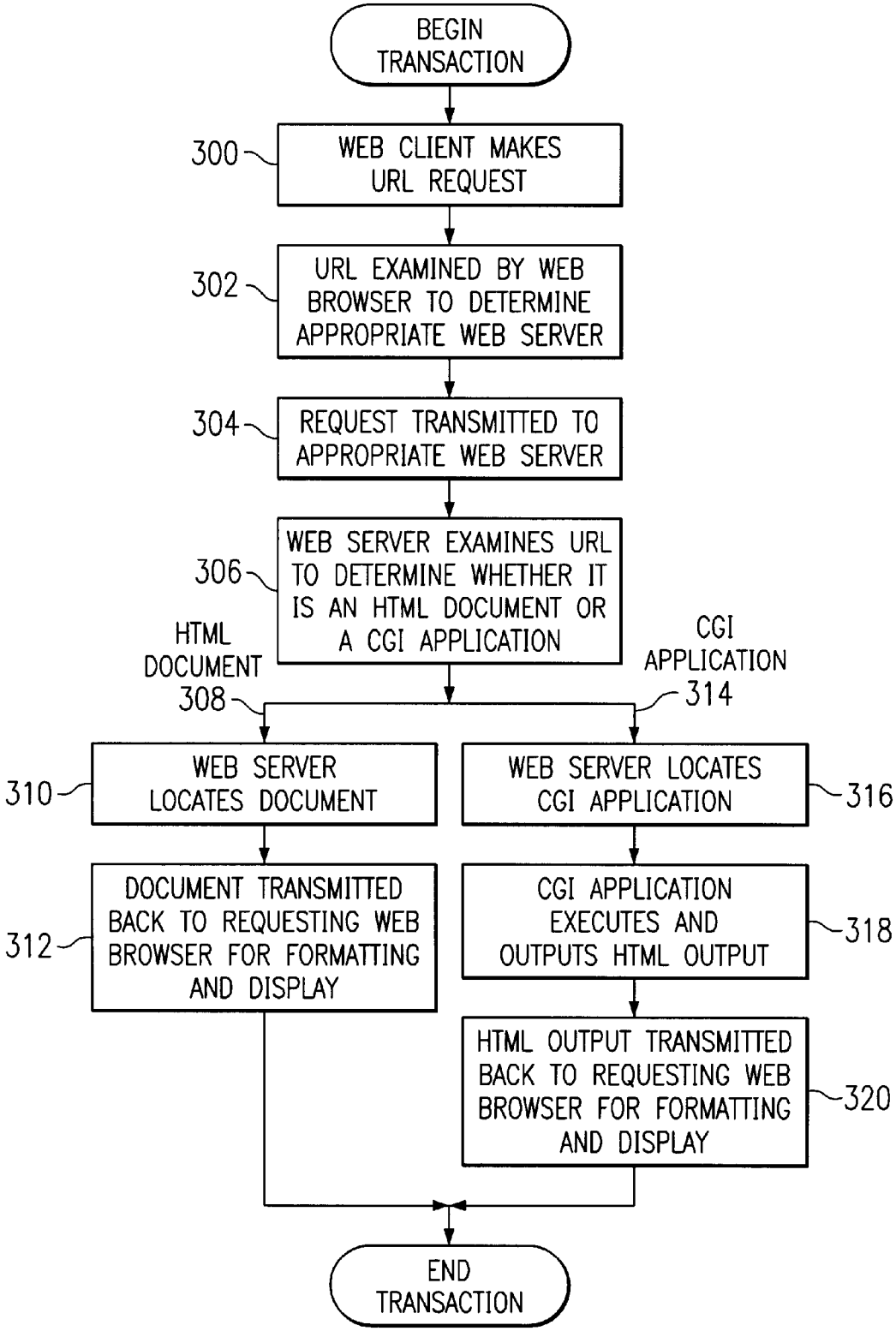


FIG. 4

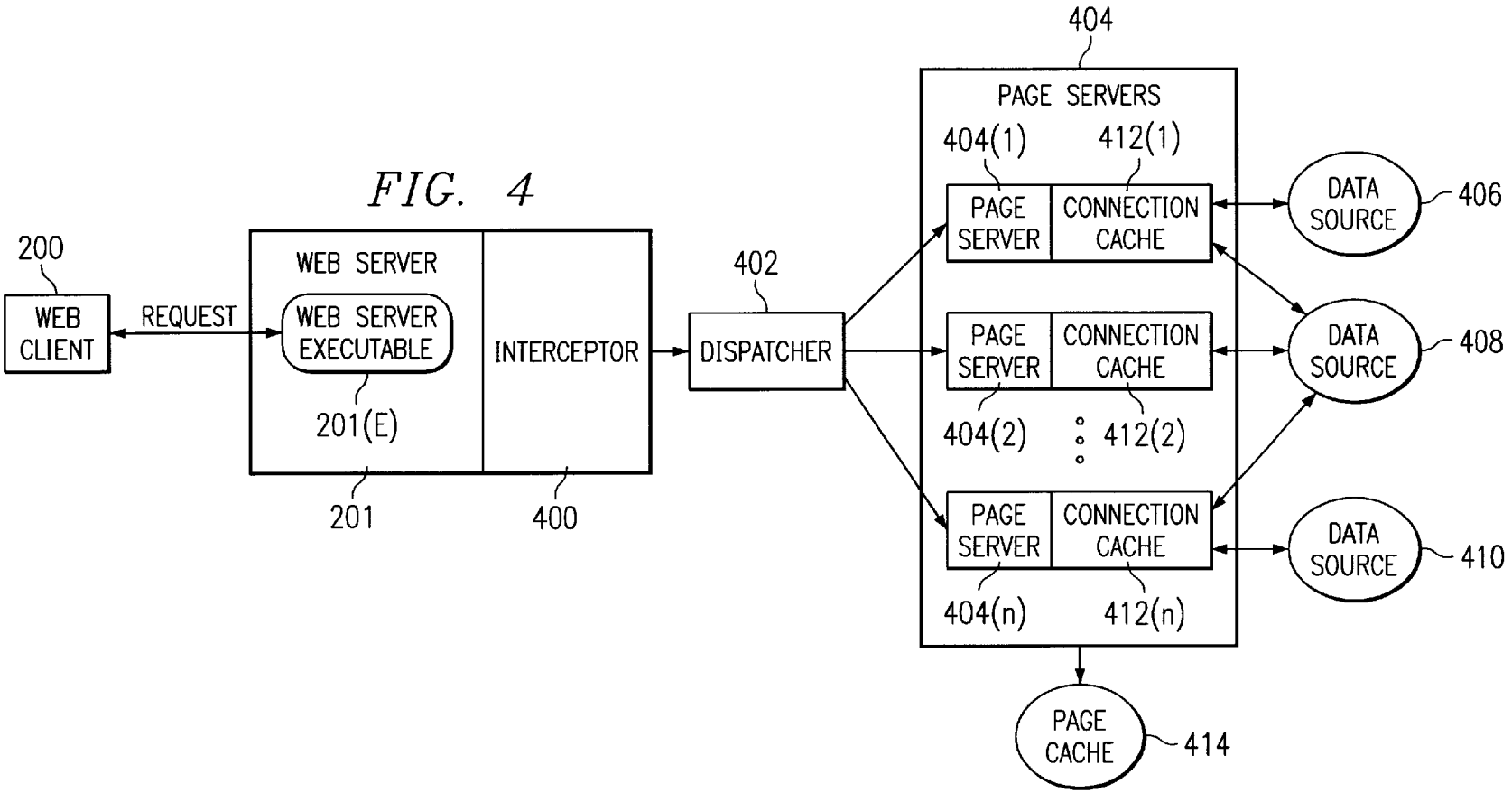
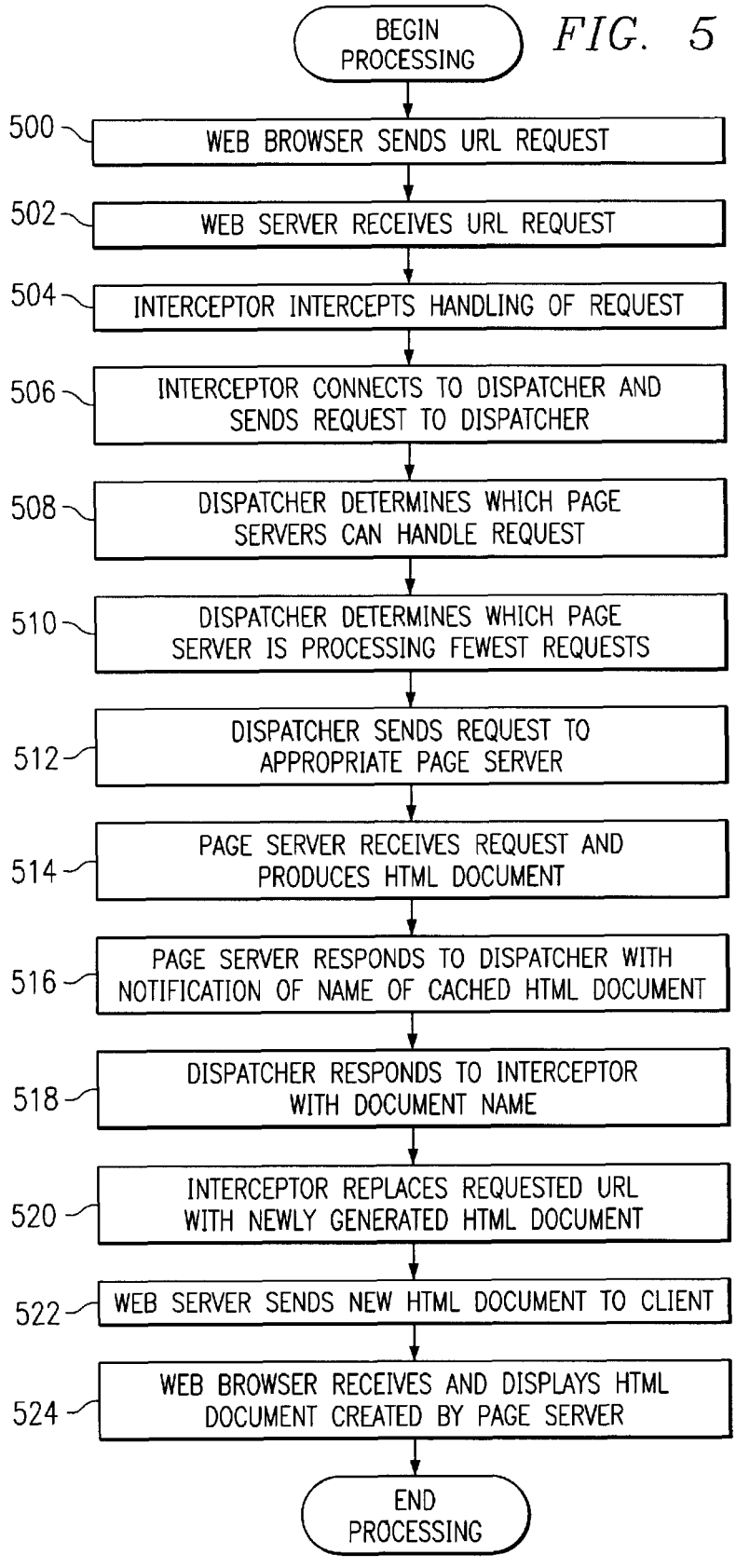

200
WEB CLIENT
REQUEST
WEB SERVER
WEB SERVER EXECUTABLE
201(E)
201
INTERCEPTOR
400
402
DISPATCHER
404
PAGE SERVERS
404(1)
412(1)
PAGE SERVER
CONNECTION CACHE
DATA SOURCE
406
PAGE SERVER
CONNECTION CACHE
DATA SOURCE
408
404(2)
412(2)
PAGE SERVER
CONNECTION CACHE
DATA SOURCE
410
404(n)
412(n)
PAGE CACHE
414

FIG. 5


BEGIN PROCESSING
500
WEB BROWSER SENDS URL REQUEST
502
WEB SERVER RECEIVES URL REQUEST
504
INTERCEPTOR INTERCEPTS HANDLING OF REQUEST
506
INTERCEPTOR CONNECTS TO DISPATCHER AND SENDS REQUEST TO DISPATCHER
508
DISPATCHER DETERMINES WHICH PAGE SERVERS CAN HANDLE REQUEST
510
DISPATCHER DETERMINES WHICH PAGE SERVER IS PROCESSING FEWEST REQUESTS
512
DISPATCHER SENDS REQUEST TO APPROPRIATE PAGE SERVER
514
PAGE SERVER RECEIVES REQUEST AND PRODUCES HTML DOCUMENT
516
PAGE SERVER RESPONDS TO DISPATCHER WITH NOTIFICATION OF NAME OF CACHED HTML DOCUMENT
518
DISPATCHER RESPONDS TO INTERCEPTOR WITH DOCUMENT NAME
520
INTERCEPTOR REPLACES REQUESTED URL WITH NEWLY GENERATED HTML DOCUMENT
522
WEB SERVER SENDS NEW HTML DOCUMENT TO CLIENT
524
WEB BROWSER RECEIVES AND DISPLAYS HTML DOCUMENT CREATED BY PAGE SERVER
END PROCESSING

SYSTEM AND METHOD FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS

This application is a division of Ser. No. 08/636,477, filed Apr. 23, 1996, now U.S. Pat. No. 5,894,554.

FIELD OF THE INVENTION

The present invention relates to the field of Internet technology. Specifically, the present invention relates to the creation and management of custom World Wide Web sites.

DESCRIPTION OF RELATED ART

The World Wide Web (the Web) represents all of the computers on the Internet that offer users access to information on the Internet via interactive documents or Web pages. These Web pages contain hypertext links that are used to connect any combination of graphics, audio, video and text, in a non-linear, non-sequential manner. Hypertext links are created using a special software language known as HyperText Mark-Up Language (HTML).

Once created, Web pages reside on the Web, on Web servers or Web sites. A Web site can contain numerous Web pages. Web client machines running Web browsers can access these Web pages at Web sites via a communications protocol known as HyperText Transport Protocol (HTTP). Web browsers are software interfaces that run on World Wide Web clients to allow access to Web sites via a simple user interface. A Web browser allows a Web client to request a particular Web page from a Web site by specifying a Uniform Resource Locator (URL). A URL is a Web address that identifies the Web page and its location on the Web. When the appropriate Web site receives the URL, the Web page corresponding to the requested URL is located, and if required, HTML output is generated. The HTML output is then sent via HTTP to the client for formatting on the client's screen.

Although Web pages and Web sites are extremely simple to create, the proliferation of Web sites on the Internet highlighted a number of problems. The scope and ability of a Web page designer to change the content of the Web page was limited by the static nature of Web pages. Once created, a Web page remained static until it was manually modified. This in turn limited the ability of Web site managers to effectively manage their Web sites.

The Common Gateway Interface (CGI) standard was developed to resolve the problem of allowing dynamic content to be included in Web pages. CGI "calls" or procedures enable applications to generate dynamically created HTML output, thus creating Web pages with dynamic content. Once created, these CGI applications do not have to be modified in order to retrieve "new" or dynamic data. Instead, when the Web page is invoked, CGI "calls" or procedures are used to dynamically retrieve the necessary data and to generate a Web page.

CGI applications also enhanced the ability of Web site administrators to manage Web sites. Administrators no longer have to constantly update static Web pages. A number of vendors have developed tools for CGI based development, to address the issue of dynamic Web page generation. Companies like Spider™ and Bluestone™, for example, have each created development tools for CGI-based Web page development. Another company, Haht Software™, has developed a Web page generation tool that uses a BASIC-like scripting language, instead of a CGI scripting language.

Tools that generate CGI applications do not, however, resolve the problem of managing numerous Web pages and requests at a Web site. For example, a single company may maintain hundreds of Web pages at their Web site. Current Web server architecture also does not allow the Web server to efficiently manage the Web page and process Web client requests. Managing these hundreds of Web pages in a coherent manner and processing all requests for access to the Web pages is thus a difficult task. Existing development tools are limited in their capabilities to facilitate dynamic Web page generation, and do not address the issue of managing Web requests or Web sites.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a method and apparatus for creating and managing custom Web sites. Specifically, the present invention claims a method and apparatus for managing dynamic web page generation requests.

In one embodiment, the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources. Other embodiments also include connection caches to the one or more data sources, page caches for each page server, and custom HTML extension templates for configuring the Web page.

Other objects, features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical computer system in which the present invention operates.

FIG. **2** illustrates a typical prior art Web server environment.

FIG. **3** illustrates a typical prior art Web server environment in the form of a flow diagram.

FIG. **4** illustrates one embodiment of the presently claimed invention.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention relates to a method and apparatus for creating and managing custom Web sites. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent to one of ordinary skill in the art, however, that these specific details need not be used to practice the present invention. In other instances, well-known structures, interfaces and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

FIG. **1** illustrates a typical computer system **100** in which the present invention operates. The preferred embodiment of

the present invention is implemented on an IBM™ Personal Computer manufactured by IBM Corporation of Armonk, New York. An alternate embodiment may be implemented on an RS/6000™ Workstation manufactured by IBM Corporation of Armonk, New York. It will be apparent to those of ordinary skill in the art that other computer system architectures may also be employed.

In general, such computer systems as illustrated by FIG. **1** comprise a bus **101** for communicating information, a processor **102** coupled with the bus **101** for processing information, main memory **103** coupled with the bus **101** for storing information and instructions for the processor **102**, a read-only memory **104** coupled with the bus **101** for storing static information and instructions for the processor **102**, a display device **105** coupled with the bus **101** for displaying information for a computer user, an input device **106** coupled with the bus **101** for communicating information and command selections to the processor **102**, and a mass storage device **107**, such as a magnetic disk and associated disk drive, coupled with the bus **101** for storing information and instructions. A data storage medium **108** containing digital information is configured to operate with mass storage device **107** to allow processor **102** access to the digital information on data storage medium **108** via bus **101**.

Processor **102** may be any of a wide variety of general purpose processors or microprocessors such as the Pentium™ microprocessor manufactured by Intel™ Corporation or the RS/6000™ processor manufactured by IBM Corporation. It will be apparent to those of ordinary skill in the art, however, that other varieties of processors may also be used in a particular computer system. Display device **105** may be a liquid crystal device, cathode ray tube (CRT), or other suitable display device. Mass storage device **107** may be a conventional hard disk drive, floppy disk drive, CD-ROM drive, or other magnetic or optical data storage device for reading and writing information stored on a hard disk, a floppy disk, a CD-ROM a magnetic tape, or other magnetic or optical data storage medium. Data storage medium **108** may be a hard disk, a floppy disk, a CD-ROM, a magnetic tape, or other magnetic or optical data storage medium.

In general, processor **102** retrieves processing instructions and data from a data storage medium **108** using mass storage device **107** and downloads this information into random access memory **103** for execution. Processor **102**, then executes an instruction stream from random access memory **103** or read-only memory **104**. Command selections and information input at input device **106** are used to direct the flow of instructions executed by processor **102**. Equivalent input device **106** may also be a pointing device such as a conventional mouse or trackball device. The results of this processing execution are then displayed on display device **105**.

The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system **100** in a conventional manner. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium **108** and subsequently loaded into and executed within computer system **100**. Once initiated, the software of the preferred embodiment operates in the manner described below.

FIG. **2** illustrates a typical prior art Web server environment. Web client **200** can make URL requests to Web server **201** or Web server **202**. Web servers **201** and **202** include Web server executables, **201**(E) and **202**(E) respectively, that perform the processing of Web client requests. Each Web server may have a number of Web pages **201**(1)–(n) and **202**(1)–(n). Depending on the URL specified by the Web client **200**, the request may be routed by either Web server executable **201**(E) to Web page **201** (1), for example, or from Web server executable **202**(E) to Web page **202** (1). Web client **200** can continue making URL requests to retrieve other Web pages. Web client **200** can also use hyperlinks within each Web page to "jump" to other Web pages or to other locations within the same Web page.

FIG. **3** illustrates this prior art Web server environment in the form of a flow diagram. In processing block **300**, the Web client makes a URL request. This URL request is examined by the Web browser to determine the appropriate Web server to route the request to in processing block **302**. In processing block **304** the request is then transmitted from the Web browser to the appropriate Web server, and in processing block **306** the Web server executable examines the URL to determine whether it is a HTML document or a CGI application. If the request is for an HTML document **308**, then the Web server executable locates the document in processing block **310**. The document is then transmitted back through the requesting Web browser for formatting and display in processing block **312**.

If the URL request is for a CGI application **314**, however, the Web server executable locates the CGI application in processing block **316**. The CGI application then executes and outputs HTML output in processing block **318** and finally, the HTML output is transmitted back to requesting Web browser for formatting and display in processing block **320**.

This prior art Web server environment does not, however, provide any mechanism for managing the Web requests or the Web sites. As Web sites grow, and as the number of Web clients and requests increase, Web site management becomes a crucial need.

For example, a large Web site may receive thousands of requests or "hits" in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running "multi-threaded" operating systems that allow transactions to be processed by independent "threads," all the threads are nevertheless on a single machine, sharing a processor. As such, the Web executable thread may hand off a request to a processing thread, but both threads will still have to be handled by the processor on the Web server machine. When numerous requests are being simultaneously processed by multiple threads on a single machine, the Web server can slow down significantly and become highly inefficient. The claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers.

FIG. **4** illustrates one embodiment of the presently claimed invention. Web client **200** issues a URL request that is processed to determined proper routing. In this embodiment, the request is routed to Web server **201**. Instead of Web server executable **201**(E) processing the URL request, however, Interceptor **400** intercepts the request and routes it to Dispatcher **402**. In one embodiment, Interceptor **400** resides on the Web server machine as an extension to Web server **201**. This embodiment is appropriate for Web servers such as Netsite™ from Netscape, that support such extensions. A number of public domain Web servers, such as NCSA™ from the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign, however, do not provide support for

this type of extension. Thus, in an alternate embodiment, Interceptor **400** is an independent module, connected via an "intermediate program" to Web server **201**. This intermediate program can be a simple CGI application program that connects Interceptor **400** to Web server **201**. Alternate intermediate programs the perform the same functionality can also be implemented.

In one embodiment of the invention, Dispatcher **402** resides on a different machine than Web server **201**. This embodiment overcomes the limitation described above, in prior art Web servers, wherein all processing is performed by the processor on a single machine. By routing the request to Dispatcher **402** residing on a different machine than the Web server executable **201**(E), the request can then be processed by a different processor than the Web server executable **201**(E). Web server executable **201**(E) is thus free to continue servicing client requests on Web server **201** while the request is processed "off-line," at the machine on which Dispatcher **402** resides.

Dispatcher **402** can, however, also reside on the same machine as the Web server. The Web site administrator has the option of configuring Dispatcher **402** on the same machine as Web server **201**, taking into account a variety of factors pertinent to a particular Web site, such as the size of the Web site, the number of Web pages and the number of hits at the Web site. Although this embodiment will not enjoy the advantage described above, namely off-loading the processing of Web requests from the Web server machine, the embodiment does allow flexibility for a small Web site to grow. For example, a small Web site administrator can use a single machine for both Dispatcher **402** and Web server **201** initially, then off-load Dispatcher **402** onto a separate machine as the Web site grows. The Web site can thus take advantage of other features of the present invention regardless of whether the site has separate machines configured as Web servers and dispatchers.

Dispatcher **402** receives the intercepted request and then dispatches the request to one of a number of Page servers **404** (1)–(n). For example, if Page server **404** (1) receives the dispatched request, it processes the request and retrieves the data from an appropriate data source, such as data source **406**, data source **408**, or data source **410**. Data sources, as used in the present application, include databases, spreadsheets, files and any other type of data repository. Page server **404** (1) can retrieve data from more than one data source and incorporate the data from these multiple data sources in a single Web page.

In one embodiment, each Page server **404**(1)–(n) resides on a separate machine on the network to distribute the processing of the request. Dispatcher **402** maintains a variety of information regarding each Page server on the network, and dispatches requests based on this information. For example, Dispatcher **402** retains dynamic information regarding the data sources that any given Page server can access. Dispatcher **402** thus examines a particular request and determines which Page servers can service the URL request. Dispatcher **402** then hands off the request to the appropriate Page server.

For example, if the URL request requires financial data from data source **408**, dispatcher **402** will first examine an information list. Dispatcher **402** may determine that Page server **404**(3), for example, has access to the requisite data in data source **408**. Dispatcher **402** will thus route the URL request to Page server **404**(3). This "connection caching" functionality is described in more detail below, under the heading "Performance." Alternately, Dispatcher **402** also has the ability to determine whether a particular Page server already has the necessary data cached in the Page server's page cache (described in more detail below, under the heading "Performance"). Dispatcher **402** may thus determine that Page server **404**(1) and **404**(2) are both logged into Data source **408**, but that Page server **404**(2) has the financial information already cached in Page server **404**(2)'s page cache. In this case, Dispatcher **402** will route the URL request to Page server **404**(2) to more efficiently process the request.

Finally, Dispatcher **402** may determine that a number or all Page servers **404**(1)–(n) are logged into Data source **408**. In this scenario, Dispatcher **402** can examine the number of requests that each Page server is servicing and route the request to the least busy page server. This "load balancing" capability can significantly increase performance at a busy Web site and is discussed in more detail below, under the heading "Scalability".

If, for example, Page server **404**(2), receives the request, Page server **404**(2) will process the request. While Page server **404**(2) is processing the request, Web server executable **201**(E) can concurrently process other Web client requests. This partitioned architecture thus allows both Page server **404**(2) and Web server executable **201**(E) to. simultaneously process different requests, thus increasing the efficiency of the Web site. Page server **404**(2) dynamically generates a Web page in response to the Web client request, and the dynamic Web page is then either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

One embodiment of the claimed invention also provides a Web page designer with HTML extensions, or "dyna" tags. These dyna tags provide customized HTML functionality to a Web page designer, to allow the designer to build customized HTML templates that specify the source and placement of retrieved data. For example, in one embodiment, a "dynatext" HTML extension tag specifies a data source and a column name to allow the HTML template to identify the data source to log into and the column name from which to retrieve data. Alternatively, "dyna-anchor" tags allow the designer to build hyperlink queries while "dynablock" tags provide the designer with the ability to iterate through blocks of data. Page servers use these HTML templates to create dynamic Web pages. Then, as described above, these dynamic Web pages are either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

The presently claimed invention provides numerous advantages over prior art Web servers, including advantages in the areas of performance, security, extensibility and scalability

Performance

One embodiment of the claimed invention utilizes connection caching and page caching to improve performance. Each Page server can be configured to maintain a cache of connections to numerous data sources. For example, as illustrated in FIG. **4**, Page server **404**(1) can retrieve data from data source **406**, data source **408** or data source **410**. Page server **404**(1) can maintain connection cache **412**(1), containing connections to each of data source **406**, data source **408** and data source **410**, thus eliminating connect times from the Page servers to those data sources.

Additionally, another embodiment of the present invention supports the caching of finished Web pages, to optimize the performance of the data source being utilized. This "page

caching" feature, illustrated in FIG. **4** as Page cache **414**, allows the Web site administrator to optimize the performance of data sources by caching Web pages that are repeatedly accessed. Once the Web page is cached, subsequent requests or "hits" will utilize the cached Web page rather than re-accessing the data source. This can radically improve the performance of the data source.

Security

The present invention allows the Web site administrator to utilize multiple levels of security to manage the Web site. In one embodiment, the Page server can utilize all standard encryption and site security features provided by the Web server. In another embodiment, the Page server can be configured to bypass connection caches **412(1)**–(n), described above, for a particular data source and to require entry of a user-supplied identification and password for the particular data source the user is trying to access.

Additionally, another embodiment of the presently claimed invention requires no real-time access of data sources. The Web page caching ability, described above, enables additional security for those sites that want to publish non-interactive content from internal information systems, but do not want real-time Internet accessibility to those internal information systems. In this instance, the Page server can act as a "replication and staging agent" and create Web pages in batches, rather than in real-time. These "replicated" Web pages are then "staged" for access at a later time, and access to the Web pages in this scenario is possible even if the Page server and dispatcher are not present later.

In yet another embodiment, the Page server can make a single pass through a Web library, and compile a Web site that exists in the traditional form of separately available files. A Web library is a collection of related Web books and Web pages. More specifically, the Web library is a hierarchical organization of Web document templates, together with all the associated data source information. Information about an entire Web site is thus contained in a single physical file, thus simplifying the problem of deploying Web sites across multiple Page servers. The process of deploying the Web site in this embodiment is essentially a simple copy of a single file.

Extensibility

One embodiment of the present invention provides the Web site administrator with Object Linking and Embedding (OLE) 2.0 extensions to extend the page creation process. These OLE 2.0 extensions also allow information submitted over the Web to be processed with user-supplied functionality. Utilizing development tools such as Visual Basic, Visual C++ or PowerBuilder that support the creation of OLE 2.0 automation, the Web site administrator can add features and modify the behavior of the Page servers described above. This extensibility allows one embodiment of the claimed invention to be incorporated with existing technology to develop an infinite number of custom web servers.

For example, OLE 2.0 extensions allow a Web site administrator to encapsulate existing business rules in an OLE 2.0 automation interface, to be accessed over the Web. One example of a business rule is the steps involved in the payoff on an installment or mortgage loan. The payoff may involve, for example, taking into account the current balance, the date and the interest accrued since the last payment. Most organizations already have this type of business rule implemented using various applications, such as Visual Basic for client-server environments, or CICS programs on mainframes. If these applications are OLE 2.0 compliant, the Page server "dynaobject" HTML extension tag can be used to encapsulated the application in an OLE 2.0 automation interface. The Page server is thus extensible, and can incorporate the existing application with the new Page server functionality.

Scalability

One embodiment of the claimed invention allows "plug and play" scalability. As described above, referring to FIG. **4**, Dispatcher **402** maintains information about all the Page servers configured to be serviced by Dispatcher **402**. Any number of Page servers can thus be "plugged" into the configuration illustrated in FIG. **4**, and the Page servers will be instantly activated as the information is dynamically updated in Dispatcher **402**. The Web site administrator can thus manage the overhead of each Page server and modify each Page server's load, as necessary, to improve performance. In this manner, each Page server will cooperate with other Page servers within a multi-server environment. Dispatcher **402** can examine the load on each Page server and route new requests according to each Page server's available resources. This "load-balancing" across multiple Page servers can significantly increase a Web site's performance.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention. A Web browser sends a URL request to a Web server in processing block **500**. In processing block **502**, the Web server receives the URL request, and an interceptor then intercepts the handling of the request in processing block **504**. The interceptor connects to a dispatcher and sends the URL request to the dispatcher in processing block **506**. In processing block **508**, the dispatcher determines which Page servers can handle the request. The dispatcher also determines which Page server is processing the fewest requests in processing block **510**, and in processing block **512**, the dispatcher sends the URL request to an appropriate Page server. The Page server receives the request and produces an HTML document in processing block **514**. The Page server then responds to the dispatcher with notification of the name of the cached HTML document in processing block **516**. In processing block **518**, the dispatcher responds to the interceptor with the document name, and the interceptor then replaces the requested URL with the newly generated HTML document in processing block **520**. The Web server then sends the new HTML document to the requesting client in processing block **522**. Finally, the Web browser receives and displays the HTML document created by the Page server at processing block **524**.

Thus, a method and apparatus for creating and managing custom Web sites is disclosed. These specific arrangements and methods described herein are merely illustrative of the principles of the present invention. Numerous modifications in form and detail may be made by those of ordinary skill in the art without departing from the scope of the present invention. Although this invention has been shown in relation to a particular preferred embodiment, it should not be considered so limited. Rather, the present invention is limited only by the scope of the appended claims.

We claim:

1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

routing a request from a Web server to a page server, said page server receiving said request and releasing said

Web server to process other requests wherein said routing step further includes the steps of:

intercepting said request at said Web server and routing said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

2. The computer-implemented method in claim **1** wherein said step of routing said request includes the steps of:

routing said request from said Web server to a dispatcher; and

dispatching said request to said page server.

3. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

4. The computer-implemented method in claim **1** wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

5. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

6. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of logging into said one or more data sources.

7. The computer-implemented method in claim **1** wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

8. The computer-implemented method in claim **1** wherein said page server includes tag-based text templates for configuring said Web page.

9. The computer-implemented method in claim **8** wherein said step of processing said request further includes the step of inserting said-dynamically retrieved data from said one or more data sources into said tag-based text templates.

10. The computer-implemented method in claim **8** wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

11. The computer-implemented method in claim **8** wherein said tag-based text templates include HTML templates.

12. The computer-implemented method in claim **1** wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

13. The computer-implemented method in claim **1** wherein said step of processing said request further includes the step of processing an object handling extension.

14. The computer-implemented method in claim **13** wherein said object handling extension is an OLE extension.

15. A computer-implemented method comprising the steps of:

transferring a request from an HTTP-compliant device to a page server, said page server receiving said request and releasing said HTTP-compliant device to process other requests wherein said transferring step further includes the steps of:

intercepting said request at said HTTP-compliant device and transferring said request to said page server;

processing said request, said processing being performed by said page server while said HTTP-compliant device concurrently processes said other requests; and

dynamically generating a page in response to said request, said page including data dynamically retrieved from one or more data sources.

16. The computer-implemented method in claim **15** wherein said step of transferring said request includes the steps of:

transferring said request from said HTTP-compliant device to a dispatcher; and

dispatching said request to said page server.

17. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

18. The computer-implemented method in claim **15** wherein said step of dynamically generating said page includes the step of dynamically retrieving said data from said one or more data sources.

19. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

20. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of logging into said one or more data sources.

21. The computer-implemented method in claim **15** wherein said step of dynamically generating said page includes the step of maintaining a page cache containing said page.

22. The computer-implemented method in claim **15** wherein said page server includes tag-based text templates for configuring said page.

23. The computer-implemented method in claim **22** wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said tag-based text templates.

24. The computer-implemented method in claim **22** wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

25. The computer-implemented method in claim **22** wherein said tag-based text templates include HTML templates.

26. The computer-implemented method in claim **15** wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

27. The computer-implemented method in claim **15** wherein said step of processing said request further includes the step of processing an object handling extension.

28. The computer-implemented method in claim **27** wherein said object handling extension is an OLE extension.

29. A computer-implemented method comprising the steps of:

transferring a request from an HTTP-compliant device to a dispatcher;

maintaining dynamic information regarding data sources a given page server may access;

dispatching said request to an appropriate page server based on said request and based on said dynamic information, said page server receiving said request and

releasing said HTTP-compliant device to process other requests;

processing said request, said processing being performed by said page server while said HTTP-compliant device concurrently processes said other requests; and

dynamically generating a page in response to said request, said page including data dynamically retrieved from one or more data sources.

* * * * *

Exhibit 3

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC., Plaintiffs, v. EPICREALM LICENSING, LP, Defendant. AND RELATED COUNTERCLAIMS	C.A. No. 06-cv-414 (SLR)

PLAINTIFFS' IDENTIFICATION OF CLAIM TERMS THAT NEED CONSTRUCTION AND PRELIMINARY CLAIM CONSTRUCTIONS

In accordance with the Court's Joint Discovery Plan and Discovery Order, Plaintiffs Oracle Corp. and Oracle U.S.A. Inc. (collectively "Oracle") make the following disclosures.

Oracle reserves the right to supplement, amend or otherwise revise its preliminary claim constructions as necessary. Particularly, Oracle submits its preliminary constructions without the benefit of complete discovery. Oracle may supplement, amend or revise its preliminary contentions in light of Defendant epicRealm Licensing LP's ("epicRealm") infringement contentions or as further evidence is discovered during the course of discovery.

Pursuant to the Court's Order, Oracle is prepared to meet and confer with epicRealm at a mutually agreeable time and place for the purpose of narrowing the claim construction issues and finalizing preparation of the Joint Claim Construction Statement.

Claim Term	Oracle's Proposed Claim Construction
appropriate page server ('335 – claim 29)	a page server determined to be capable of processing said request based on said dynamic information

Claim Term	Oracle's Proposed Claim Construction
computer system ('554 - claims 9, 11)	**first computer system** = a combination of computer hardware and software, under control of a single party, capable of generating a dynamic Web page generation request **second computer system** = a combination of computer hardware and software, under control of a single party, and distinct from said first computer system, capable of receiving a dynamic Web page generation request over a network from said first computer system
computer-implemented method for managing a dynamic Web page generation request to a Web server ('554 - claim 1) ('335 - claim 1)	a method at least partially involving a computer for processing a dynamic Web page generation request sent to a Web server
connection cache ('554 – claim 4) ('335 – claims 5, 19)	a store of information identifying open connections to data sources for eliminating subsequent connect times to those data sources
custom HTML extension templates ('554 – claims 7, 8)	custom files using non-HTML markers for specifying where data from data sources should be inserted to create an HTML file
data dynamically retrieved ('554 – claims 1, 9) ('335 – claims 1, 10, 15, 24, 29)	data retrieved in response to a request at the time it is needed
data sources ('554 – claims 1-5, 8-9, 11) ('335 – claims 1, 3-6, 9-10, 12, 15, 17-20, 23-24, 26 29)	computerized repositories from which data may be retrieved electronically
dispatcher ('554 – 1, 9, 11) ('335 – claims 2, 16, 29)	a software program for determining which page server should be used to process a dynamic web page generation request
dispatching said request to said page server ('554 – claims 1, 9, 11) ('335 – claims 2, 16, 29)	analyzing a request to make an informed selection of which page server should process the request, and sending the request to that page server
dynamic Web page ('554 – claims 1, 9, 11) ('335 – claim 1)	a Web page whose content changes over time and therefore cannot be stored statically

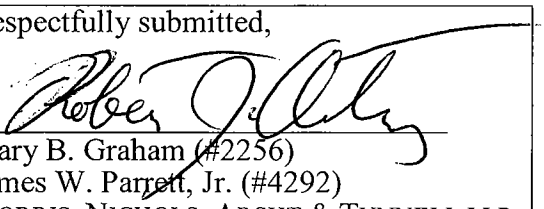
Claim Term	Oracle's Proposed Claim Construction
dynamic Web page generation request ('554 – claims 1, 9, 11) ('335 - claim 1)	a request for generation of a dynamic Web page
dynamically generating a Web page ('554 – 1, 3, 6, 9, 11) ('335 – 1, 4, 7, 15, 18, 21, 29)	creating on demand a Web page that did not previously exist
dynamically retrieving said data ('554 – claim 3) ('335 – claims 4, 18)	retrieving data in response to a request at the time it is needed
HTTP-compliant device ('335 – claims 15, 16, 29)	a device running software that implements the Hypertext Transfer Protocol
identifying said one or more data sources ('554 – claim 2)	listing in an HTML extension tag the location of said one or more data sources
inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates ('554 - claim 8)	replacing tags in said custom HTML extension templates with data dynamically retrieved from said one or more data sources
intercepting said request at [said Web server/ second computer system/HTTP-compliant device] ('554 - claims 1, 9, 10, 11) ('335 - claims 1, 15)	receiving a request at the [Web server machine/second computer system/HTTP-compliant device] and diverting the request before the [Web server/second computer system/HTTP-compliant device] executable can process the request
interceptor ('554 - claim 10)	a software component that performs intercepting
logging into ('554 - claim 5)	presenting credentials that allow access to
machine readable medium ('554 - claim 11)	a hard disk, floppy disk, CD-ROM, magnetic tape, or other magnetic or optical data storage medium

Claim Term	Oracle's Proposed Claim Construction
maintaining a connection cache to said one or more data sources ('554 - claim 4) **maintaining** a page cache containing said Web page ('554 - claim 6)	storing and updating
maintaining dynamic information ('335 – claim 29)	storing and updating information that changes over time
managing A computer-implemented method for **managing** a dynamic Web page generation request to a Web server ('554 - claims 1, 9) ('335 - claim 1)	processing
Means for generating said request ('554 – claim 9)	112 ¶6 corresponding function: generating said request structure: A processor of a computer that is, or has, a Web client running a Web browser
means for receiving said request from said first computer ('554 – claim 9)	112 ¶6 corresponding function: receiving said request from said first computer structure: A processor of a computer that is, or has, a Web server running Web server executable
networked system a **networked system** for managing a dynamic Web page generation request ('554 – claim 9)	a system of computers, interconnected by a network, which interacts with the World Wide Web
other requests ('554 – claims 1, 9, 11) ('335 – claims 1, 15, 29)	requests other than said request
page cache ('554 - claim 6) ('335 – claims 7, 21)	a memory that stores finished Web pages which the page server uses to fulfill requests instead of re-accessing the one or more data sources

Claim Term	Oracle's Proposed Claim Construction
page server ('554 patent - claims 1, 4, 7, 9, 10, 11) ('335 patent - claims 1, 2, 5, 8, 15, 16, 19, 22, 29)	a server machine, distinct from the [Web server machine/second computer system/HTTP-compliant device], which generates dynamic Web pages
page server processing means ('554 – claim 9)	112 ¶6 corresponding function: processing structure: a computer programmed to process a dynamic Web page generation request by retrieving data dynamically from one or more data sources and dynamically generating a Web page responsive to said request
request ('554 – all) ('355 – all)	A message that asks for a Web page specified by a URL
router ('554 - claims 9, 10)	a device that intercepts and dispatches dynamic Web page generation requests
routing ('554 - claims 1, 9, 10, 11) ('335 - claims 1, 2)	transmitting
said page server receiving said request and releasing said Web server [second computer system or HTTP-compliant device] to process other requests ('554 – claims 1, 9, 11) ('335 – claims 1, 15, 29)	said page server receiving said request and said page server performing an act (separate from merely receiving the request) that expressly communicates to said [Web server/second computer system/HTTP-compliant device] that it may now resume processing other requests
said processing being performed by said page server while said [Web server/HTTP-compliant device] concurrently processes said other requests ('554 – claims 1, 11) ('335 – claims 1, 15, 29)	said page server generating the requested dynamic Web page while said [Web server/HTTP-compliant device] processes other requests literally at the same time
transferring ('335 – claims 15-16, 29)	sending
Web page ('554 - claims 1, 3, 6, 7, 9, 11) ('335 - 1, 4, 7, 8)	a file containing embedded commands in a Web formatting language such as HTML, capable of being displayed on a Web browser

Claim Term	Oracle's Proposed Claim Construction
Web server ('554 - claims 1, 11) ('335 - 1, 2)	a computer running HTTP-compliant server software which receives Web page requests and returns Web pages in response to the requests

Respectfully submitted,



Mary B. Graham (#2256)
James W. Parrett, Jr. (#4292)
MORRIS, NICHOLS, ARSHT & TUNNELL LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
302.658.9200

James G. Gilliland
Theodore T. Herhold
Chad E. King
Robert J. Artuz
Eric M. Hutchins
TOWNSEND AND TOWNSEND AND CREW LLP
379 Lytton Avenue
Palo Alto, CA 94301
650.326.2400

Attorneys for Plaintiffs
Oracle Corporation and Oracle U.S.A. Inc.

Dated: December 14, 2007

61235117 v1

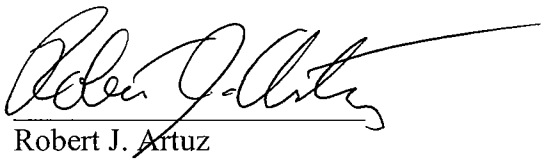
CERTIFICATE OF SERVICE

I, Robert J. Artuz, hereby certify that on December 14, 2007 I caused the foregoing to be served on the following individuals by E-Mail, as indicated:

BY E-MAIL AND U. S. MAIL

Richard L. Horwitz
David Ellis Moore
Potter Anderson & Corroon LLP
1313 N. Market St. Hercules Plaza 6th Flr
Wilmington DE 19899-0951
rhorwitz@potteranderson.com
dmoore@potteranderson.com

George Bosy
Patrcik Patras
Benjamin Bradford
Jenner & Block LLP
330 N. Wabash Avenue
Chicago, IL 60611-7603
gbosy@jenner.com
ppatras@jenner.com
bbradford@jenner.com



Robert J. Artuz

Exhibit 4

IN THE UNITED STATES DISTRICT COURT
OF THE EASTERN DISTRICT OF TEXAS
TEXARKANA DIVISION

EPICREALM, LICENSING, LLC	§	
	§	
v.	§	**Civil Action No. 5:07-CV-125-DF**
	§	
AUTOFLEX LEASING, INC., ET AL.	§	
	§	
	§	
EPICREALM, LICENSING, LLC	§	
	§	
v.	§	**Civil Action No. 5:07-CV-126-DF**
	§	
FRANKLIN COVEY CO., ET AL.	§	
	§	
	§	
EPICREALM, LICENSING, LLC	§	
	§	**Civil Action No. 5:07-CV-135-DF**
v.	§	
	§	
VARIOUS, INC.	§	

O R D E R

Before the Court is Defendants' Joint Supplemental Brief Regarding Claim Construction and in Support of Defendants' Motions for Summary Judgment of Non-infringement. Dkt. No. 659, Case No. 5:07-cv-125; Dkt. No. 601, Case No. 5:07-cv-126; Dkt. No. 271, Case No. 5:07-cv-135. Also before the Court is Plaintiffs' epicRealm Licensing, LP and Parallel Network's Brief Regarding the Proper Construction of the Releasing Limitation. Dkt. No. 660, Case No. 5:07-cv-125; Dkt. No. 602, Case No. 5:07-cv-126; Dkt. No. 272, Case No. 5:07-cv-135. The Court, having reviewed the Magistrate Judge's July 15, 2008 Report and Recommendation and the parties' briefs, issues the following Order addressing claim construction issues related to the scope and meaning of the "releasing" term previously construed by the Court.

I. BACKGROUND

The remaining defendants in the above-referenced cases, FriendFinder Network, Inc. ("FriendFinder"), Various, Inc. ("Various"), and Herbalife International of America, Inc. ("Herbalife")(collectively "Defendants"), filed motions for summary judgement of non-infringement, asserting that their accused web sites do not satisfy the "releasing" limitation as construed by this Court. On July 15, 2008, the Magistrate Judge issued a Report and Recommendation, which has been adopted as the findings and conclusions of the Court, wherein she denied without prejudice Defendants' motions. *See* Dkt. No. 616[1]. The Magistrate Judge noted that proper consideration of the pending non-infringement summary judgment motions hinges upon the Court's resolution of the conflicting claim construction positions on the "releasing" limitation advocated for the first time in the briefing on the summary judgment motions. *Id.*

Specifically, the Magistrate Judge asked the parties to address three claim construction issues relevant to Defendants' motions for summary judgment of non-infringement and which had not been addressed earlier. The three issues were identified as follows: (1) whether the operating software can be considered part of the "page server"; (2) whether "releasing" or "freeing" should be construed to merely require releasing resources or allow the Web server to handle other requests that would not otherwise be handled; and (3) whether the "release" of a Web server may occur before a page server receives a request (or even before a Web server receives a request). *Id.* at 15-16.

All three issues concern the claim limitation which is the subject of Defendants' summary judgment motions, namely "said page server receiving said request and releasing said

[1] Citations to docket entries are hereinafter only provided as docketed in Case No. 5:07-cv-125.

Web server to process other requests." The Court construed this term to mean "said page server receiving said request and said page server performing an act (separate from merely receiving the request) to free the Web server to process other requests." Dkt. No. 194 at 29. In addition to the term "page server," which the Court construed to mean "page-generating software that generates a dynamic Web page," another claim term relevant to the pending motion is "Web server." *Id.* at 14. The Court construed "Web server" to mean "software, or a machine having software, that receives Web page requests and returns Web pages in response to the requests." *Id.* at 16.

II. DISCUSSION

A. Whether the operating software can be considered part of the "page server"

The first claim construction issue identified by the Magistrate Judge is "whether the operating software can be considered part of the 'page server.'" Dkt. No. 616 at 15. After extensive briefing and oral argument from the parties, the Court construed "page server" as "page-generating software that generates a dynamic Web page." Dkt. No. 194 at 14. Plaintiffs now advocate a construction of "page server" in which the page server is not limited to the application software but further includes the operating system. Dkt. No. 660 at 8-14. Defendants claim that the proper construction of "page server" should not include the operating system. Dkt. No. 659 at 2-8.

1. epicRealm's statements at the Claim Construction Hearing

As the Magistrate Judge has recognized, epicRealm agreed during oral argument on claim construction that the "page server" claim element should be construed as "page generating software" and not as a "processing system" as earlier advocated by epicRealm. Dkt. No. 190 at 32:13 -33:7. Defendants assert that the Plaintiff's counsel's statements at the claim construction

hearing leave no doubt that epicRealm gave up hardware and the operating system software running on that hardware as being part of the claimed "page server." Dkt. No. 659 *citing* Dkt. No. 190 at 32:13-33:30. The Court is not convinced, based on the cited exchange, that Plaintiff clearly and unambiguously disavowed a claim scope excluding operating system software being part of the page server. *See* Dkt. No. 190 at 33:17-18 ("The processing system is – it's a software module which may be executed on a computer system."). Although epicRealm agreed to a construction of "page server" that included only software as opposed to hardware, epicRealm did not clearly agree to exclude any particular portions of software.

2. Intrinsic evidence and claim construction

The court proceeds to address whether Plaintiff's current attempt to construe the "page server" term to cover the operating system software conflicts with the intrinsic evidence and the claim construction order.

The Specification describes the preferred embodiment as follows:

> The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system 100 in a conventional manner. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium 108 and subsequently loaded into and executed within computer system 100. Once initiated, the software of the preferred embodiment operates in the manner described below.

('554 patent, 3:55-63).

Defendants point out that epicRealm relied on this passage during claim construction, and argue that "page server" should be limited to application software modules. Plaintiffs respond that "there is *absolutely no indication* that this 'application software' is the *only* software used in the 'present invention.'" Dkt. No. 660 at 12. Plaintiffs go on to cite to the following portion of the

Specification to argue that the Specification supports a reading that the page server may include operating system software:

> For example, a large Web site may receive thousands of requests or 'hits' in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running 'multithreaded' *operating systems that allow transactions to be processed* by independent 'threads,' all the threads are nevertheless on a single machine, sharing a processor.

('554 patent, 4:38-45)(emphasis in original).

The Court finds that the latter passage quoted by Plaintiffs does not support the position that operating system software may be part of a page server. As Defendants argue, the term "operating system" is only mentioned "in the context of shortcomings associated with the prior art." *See* Dkt. No. 659 at 4; *See also* '554 Patent, 4:38-53[2]. However, the Court also finds that the former passage describing the preferred embodiment does not limit "page server" to an application software module. Such a construction would have the result of importing limitations from the specification into the claims. *See Phillips v. AWH Corp.*, 415 F.3d at 1323; *See also Verizon Services Corp. v. Vonage Holdings Corp.*, 503 F.3d 1295, 1303 (Fed. Cir. 2007) *quoting Texas Instruments, Inc. v. U.S. Int'l Trade Comm'n*, 805 F.2d 1558, 1563 (Fed. Cir. 1986) ("This court has cautioned against limiting the claimed limitation to preferred embodiments or specific examples in the specification.").

---

[2] "Although these machines may be running 'multithreaded' *operating systems that allow transactions to be processed* by independent 'threads,' all the threads are nevertheless on a single machine, sharing a processor . . . . The claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers."

However, to the extent that Plaintiff argues that operating system software can be "part of the web server and . . . part of the page server," the Court finds that Plaintiff's position conflicts with the Court's claim construction order.

During claim construction, Plaintiffs argued that the construction of "page server" should allow the page server to run on the same machine as the Web server. Dkt. No. 194 at 12-14. In connection with this argument, epicRealm emphasized that "the specification as a whole describes a partitioned software architecture and the benefits of dividing labor between different modules." Dkt. No. 190 at 30-40. In adopting epicRealm's postion, the Court found that the specification does not exclude "the use of a partitioned software architecture on a single machine." Dkt. No. 194 at 14. Thus, pursuant to the Court's claim construction, the web server and page server can reside on the same machine. *Id.*

As Defendants argue, Plaintiffs' position that the operating system software can be a part of the web server and a part of the page server conflicts with the construction. Dkt. No. 660 at 6-8. Specifically, the Court is not convinced that Plaintiffs' proposed construction conforms with the partitioned architecture and the different modules that epicRealm originally stated characterized the specification as a whole. If the "web server" and the "page server" exist on the same machine and both include an operating system which "is a part of the web server and . . . part of the page server" as Plaintiffs argue, there would be an overlap at the operating system level between the web server and the page server. Dkt. No. 660 at 12; Dkt. No. 659 at 7. The Court agrees that this would extinguish a "meaningful distinction between two separate elements of the claims" and would be incapable of meeting the "routing" limitation ("routing said from

said Web server to a page server")[3]. *See* Dkt. No. 660 at 12; Dkt. No. 659 at 6-8; *See also* '554 Patent, 8:66.

The Court notes that Plaintiffs also state that the "partitioned software architecture" asserted by epicRealm during claim construction was vertical partitioning between functions where "[b]oth sides of the partitioning include an operating system." Dkt. No. 660 at 13. Assuming that Plaintiffs were conceptualizing a complete vertical partitioning where the web server and page server are on opposite sides of the partitioning that each include an operating system, the Court finds that such an architecture would provide a "meaningful distinction between two separate elements of the claims" and also be able to meet the "routing" limitation. *See* Dkt. No. 659 at 12.

Based on this theory, the Court declines to limit "page server" to exclude operating system software from the construction. Thus, the Court finds that no further construction is necessary at this time. Consistent with the requirement of *O2 Micro* that the Court, not the jury, address an actual dispute regarding the proper scope of the claims, the Court expressly rejects Defendants' limitation of the "page server" to exclude operating system software. *See O2 Micro International Ltd v. Beyond Innovation Technology Co.*, 521 F.3d 1351, 1362 (Fed. Cir. 2008).

B. Whether "releasing" or "freeing" should be construed to merely require releasing resources or allow the Web server to handle other requests that would not otherwise be handled

The second claim construction issue identified by the Magistrate Judge is "whether 'releasing' or 'freeing' should be construed to merely require releasing resources or allow the Web server to handle other requests that would not otherwise be handled." Dkt. No. 616 at 15-

[3] As Defendants argue, in such a situation "the Web server would effectively be 'routing' requests from itself to itself." Dkt. No. 659 at 8.

16. Plaintiffs assert that the Court's construction allows the page server to perform an act "to free resources on the Web server to process other requests more quickly." Dkt. No. 660 at 3-8. Defendants argue that the Court's construction requires that the "act" free the Web Server to process new requests. Dkt. No. 659 at 8-12.

The Court first looks to the language of the claims. *See Phillips v. AWH Corp.,* 415 F.3d at 1314 ("Quite apart from the written description and the prosecution history, the claims themselves provide substantial guidance as to the meaning of particular claim terms."). The language of the claim provides that the Page server "releas[es] [the] Web server to process other requests." '554 Patent, 8:67-9:1 & 10:30-33. As Defendants argue, the language of the claims recites that "said Web server" is released "to process other requests," not to process requests more quickly. *See* Dkt. No. 659 at 9.

However, the Specification provides some background on the Patent's concept of off-loading requests from the Web server. The Plaintiff cites to the following parts of the Specification:

> For example, a large Web site may receive thousands of requests or 'hits' in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running 'multithreaded' operating systems that allow transactions to be processed by independent 'threads,' all the threads are nevertheless on a single machine, sharing a processor. As such, the Web executable thread may hand off a request to a processing thread, but *both threads will still have to be handled by the processor on the Web server machine*. When numerous requests are being simultaneously processed by multiple threads on a single machine, the Web server can slow down significantly and become highly *inefficient*.

('554 Patent, 4:38-53)(emphasis added).

> This embodiment overcomes the limitation described above, in prior art Web servers, routing the request to Dispatcher 402 residing on a different machine than the Web server executable 201(E), the request can then be processed by a different processor than the Web server executable 201(E). Web server

> executable 201(E) is thus *free to continue servicing* client requests on Web server 201 while the request is processed 'off-line,' at the machine on which Dispatcher 402 resides.

('554 Patent, 5:9-19)(emphasis added).

> If, for example, Page server 404(2), receives the request, Page server 404(2) will process the request. While Page server 404(2) is processing the request, Web server executable 201(E) can *concurrently process* other Web client requests. This partitioned architecture thus allows both Page server 404(2) and Web server executable 201(E) to *simultaneously process different requests*, thus increasing the efficiency of the Web site.

('554 Patent, 6:20-27)(emphasis added); *See also* Dkt. No. 660 at 4-5.

The Specification also provides that the Dispatcher 402 can "off-load[s] the processing of Web requests from the Web server machine" and "hand[] off the request to the appropriate Page server." *See* '554 Patent, 27-28 & 57-58. Thus, the Specification describes a Page server that processes requests from the Web server so that the Web server's processing resources are free to process other requests.

Plaintiffs argue that, based on the Specification, "freeing the web server to process other request" includes freeing of both the constraints on the processing resources and the number of outstanding requests. Dkt. No. 660 at 6-7 *citing* '554 Patent, 5:15-17 ("Web server executable 201(E) is thus *free to continue servicing* client requests on Web server 201 while the request is processed 'off-line,' at the machine on which Dispatcher 402 resides.") & '554 Patent, 6:24-27 ("This partitioned architecture thus allows both Page server 404(2) and Web server executable 201(E) to simultaneously *process different requests*, thus increasing the efficiency of the Web site.") (emphasis added). Plaintiffs also note that the parties previously agreed to the construction of "other requests" to mean "different requests" along with the above Specification to argue that "other requests" include both new and pre-existing requests. *Id.* at 7-8. Based on

the agreed construction of "other requests" and the Specification, the Plaintiff asserts that "releasing" is not limited to acts that free the web server to process new requests. *Id.*; Dkt. No. 142 at 2.

As urged by the Plaintiffs, the language "to continue servicing client requests on Web server" suggests that client requests are already being serviced on the Web server. *See* Dkt. No. 660 at 8. It follows that "free[ing] [the web server] to continue servicing client requests" would involve freeing the Web server to process pre-existing requests. Moreover, the Specification merely provides that the off-loading of requests from the web server to the page server "increas[es] the efficiency of the Web site"—a benefit that can also be provided by freeing processing resources. *See* Dkt. No. 660 at 8.

Thus, the Court agrees with the Plaintiff that "other requests" should include both new and pre-existing requests, and that "freeing the web server to process other requests" should not be limited to freeing the web server so that it may handle other requests that would not otherwise be handled, but should also include freeing the server so that it can process other pre-existing requests—i.e., freeing the web server to "processing resources." The Court finds that Defendants' proposal that "freeing" a Web server happens only when a Web server is able to handle a *new* request (for example, when a "socket connection" or a "child process" becomes available) is too limiting.

The Court finds unpersuasive Defendants' argument that Plaintiffs' argument conflicts with the construction of "web server" because the "freeing up of memory or processor time that epicRealm has asserted for this limitation would apply only to a machine, not a software." *See* Dkt. No. 659 at 9-10. Additionally, although Defendants argue that the patentee's statement to

overcome the Examiner's rejection based on U.S. Patent No. 5,752,246 ("Rogers")[4] exclude Plaintiffs' position, the Court finds that epicRealm's statement in the prosecution history relied upon by Defendants does not clearly address the issue before the Court and thus does not constitute a prosecution disclaimer or disavowal of claim scope that limits the term "releasing" to require that the web server handle new requests. *See* Dkt. No. 659 at 10-11.

The Court finds no further construction necessary. Consistent with the requirement of *O2 Micro*, that the Court, not the jury, address an actual dispute regarding the proper scope of the claims, the Court expressly rejects Defendants' limitation of the "releasing" or "freeing" to only allowing the Web server to handle new requests that would not otherwise be handled. *See O2 Micro International Ltd v. Beyond Innovation Technology Co.*, 521 F.3d at 1362.

C. Whether the "release" of a Web server may occur before a page server/web server receives a request

The third claim construction issue identified by the Magistrate Judge is "whether the 'release' of a Web server may occur before a page server receives a request (or even before a Web server receives a request)." Dkt. No. 616 at 16. This issue arises in the context of Plaintiffs' socket-bind-listen ("SBL") calls infringement theory. *Id.* at 11-14. Since SBL calls establish a port through which the application software can receive and send messages using the operating system at boot-up or when the page server software is first invoked, this "act" is performed before any request is sent by the web server to the page server or even before the web server receives a request. *See* Dkt. No. 616 at 12. Defendants argue that an act of releasing

[4] In response to the Examiner's rejection of the claim limitation "the page server receiving the request and releasing the Web server to process other requests" based on Rogers that "taught the use of TCP/IP as the underlying protocol for all network communications," the patentee stated "at no time does Rogers teach or suggest 'concurrently' processing other requests or 'releasing said Web server to process other requests' because merely retrieving data from multiple sources does not teach or suggest these elements." Response to Office Action at 9; Office Action at 2.

cannot occur before the page server or web server receives a request since this would violate the "bedrock principles" of claim construction that claim terms must be construed in the context of the surrounding words of the claim and that each claim term be given effect. Dkt. No. 659 at 13 Plaintiffs argue that "no specific requirements should tie this act to act to any particular time." Dkt. No. 660 at 14.

The Court agrees with the Defendant that "claim term[s] [should be construed] in the context of the particular claim in which the disputed term appears." *See Phillips v. AWH Corp.*, 415 F.3d at 1313; *See also ACTV, Inc. v. Walt Disney Co.*, 346 F.3d 1082, 1088 (Fed. Cir. 2003) ("First and foremost, the analytical focus of claim construction must begin, and remain centered, on the language of the claims themselves."). Claim 1 of the '554 Patent provides as follows:

> 1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of :
> routing said request from said Web server to a page server.
> said page server receiving said request and releasing said Web server to process other requests . . .

'554 Patent, 8:63-9:1. The term "said page server receiving said request and releasing said web server to process other requests" includes two separate acts: (1) the page server receiving a said request; and (2) the page server releasing the web server to process other requests. As explained by the Defendant, "said request" in the term refers to "a dynamic Web page generation request" that is routed from the web server to the page server and "other requests" refer to "requests different from said request". *See* Dkt. No. 659 at 13-14. Thus, in order for the page server to release the web server to process "other requests"–i.e. requests different from the dynamic web page generation request routed from the web server to the page server, the page server must have received the dynamic web page generation request from the web server. *TALtech Ltd. v. Esquel Apparel, Inc.*, 2008 WL 2165996, *4 (Fed. Cir. May 22, 2008) *citing Altiris, Inc. v. Symantec*

Corp., 318 F.3d 1363, 1369-70 (Fed. Cir. 2003) *citing Interactive Gift Express, Inc. v. Compuserve, Inc.*, 256 F.3d 1323, 1343 (Fed. Cir. 2001) ("the claim requires an ordering of steps when the claim language, as a matter of logic or grammar, requires that the steps be performed in the order written, or the specification directly or implicitly requires such a narrow construction.").

Furthermore, as Defendants argue, Plaintiffs' interpretation would have the effect of obviating the need for a web server. *See* Dkt. No. 659 at 14. The Court finds that such a reading should be discouraged. *Mangosoft, Inc. v. Oracle Corp.*, 525 F.3d 1327, 1331 (Fed. Cir. 2008) *quoting Merck & Co. v. Teva Pharm, USA, Inc.*, 395 F.3d 1364, 1372 (Fed. Cir. 2005) ("A claim construction that gives meaning to all the terms of the claim is preferred over one that does not do so.").

Although Plaintiff quotes a portion of the Specification to support their argument that the web server may be freed before the page server receives the request[5], the Court finds that the quoted passage simply describes an embodiment where first, a dynamic page request is routed from the web server to the page server; second, the page server releases the web server by processing the request itself; and third, the web server then is free to continue processing other requests. *See* Dkt. No. 660 at 14-15.

Thus, in light of the intrinsic record as a whole, the Court holds that "releasing" may not be performed prior to the receiving of the request. The Court finds no further construction necessary. Consistent with the requirement of *O2 Micro* that the Court, not the jury, address an

---

[5] By routing the request to Dispatcher 402 residing on a different machine than the Web server executable 201(E), the request can then be processed by a different processor than the Web server executable 201(E). *Web server executable 201(E) is thus free to continue servicing client requests on Web server 201 while the request is processed 'off-line,' at the machine on which Dispatcher 402 resides*. '554 Patent, 5:12-19 (emphasis added).

actual dispute regarding the proper scope of the claims, the Court expressly rejects Plaintiffs' assertion that the "releasing" can occur before the step of "receiving said request." *See O2 Micro International Ltd v. Beyond Innovation Technology Co.*, 521 F.3d at 1362.

III. CONCLUSION

For the foregoing reasons, the Court finds that no further construction of the term, "said page server receiving said request and releasing said Web server to process other requests," is necessary at this time. The Court, however, notes that (1) the construction of "page server" does not exclude operating system software; (2) "releasing" or "freeing" need not allow the web server to handle other requests that would not otherwise be handled; and (3) the web server may not be "releas[ed]" before a page server or web server receives a request.

The Court hereby **GRANTS** Defendants leave to renew their summary judgment motions upon entry of this Order.

IT IS SO ORDERED.

SIGNED this 7th day of August, 2008.

DAVID FOLSOM
UNITED STATES DISTRICT JUDGE

Exhibit 5

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 6

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 7

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 8

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 9

IN THE UNITED STATES DISTRICT COURT FOR THE
WESTERN DISTRICT OF PENNSYLVANIA

- - - -

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs,	)	
	)	C.A. No. 06-cv-414
-vs-	)	(SLR)
	)	
EPICREALM LICENSING, LP,	)	
Defendant.	)	

- - - -

VIDEOTAPED DEPOSITION OF:
MICHAEL SHAMOS, Ph.D., J.D.

- - - -

DATE: June 30, 2008
Monday, 9:12 a.m.

LOCATION: JONES DAY
One Mellon, 31st Floor
500 Grant Street
Pittsburgh, PA 15219

TAKEN BY: Defendant

REPORTED BY: Heidi H. Willis, RPR, CRR
Notary Public
Reference No. HW07873

VIDEOTAPED DEPOSITION OF MICHAEL SHAMOS, Ph.D., J.D., a witness, called by the Defendant for examination, in accordance with the Federal Rules of Civil Procedure, taken by and before Heidi H. Willis, RPR, CRR, a Court Reporter and Notary Public in and for the Commonwealth of Pennsylvania, at the offices of Jones Day, One Mellon, 31st Floor, 500 Grant Street, Pittsburgh, Pennsylvania, on Monday, June 30, 2008, commencing at 9:12 a.m.

- - - -

APPEARANCES:

FOR THE PLAINTIFFS:
Robert J. Artuz, Esq.
rjartuz@townsend.com
TOWNSEND and TOWNSEND and CREW, LLP
379 Lytton Avenue
Palo Alto, CA 94301-1431
P 650-326-2400
F 650-326-2422

FOR THE DEFENDANT:
Patrick L. Patras, Esq.
ppatras@jenner.com
Benjamin Bradford, Esq.
bbradford@jenner.com
JENNER & BLOCK, LLP
One IBM Plaza
Chicago, IL 60611-7603
P 312-222-9350
F 312-222-2945

FOR THE TEXAS DEFENDANTS FRED FINDER NETWORK and VARIOUS, INC.:
Hector J. Ribera, Esq.
hribera@fenwick.com
FENWICK & WEST, LLP
Silicon Valley Center
801 California Street
Mountain View, CA 94041
P 650-988-8500
F 650-938-5200

APPEARANCES (CONT'D):

FOR THE TEXAS DEFENDANT HERBALIFE INTERNATIONAL OF AMERICAS, INC.:
Andrey Belenky, Esq.
abelenky@jonesday.com
JONES DAY
222 East 41st Street
New York, NY 10017-6702
P 212-326-3939
F 212-755-7306

FOR PARALLEL NETWORKS and EPICREALM:
Kevin J. Meek, Esq.
kevin.meek@bakerbotts.com
BAKER BOTTS, LLP
200 Ross Avenue
Dallas, TX 75201-2980
P 214-953-6500
F 214-953-6503
(Appearing Telephonically)

ALSO PRESENT:
Matthew Sarboraria, Esq.
matthew.sarboraria@oracle.com
Oracle
500 Oracle Parkway
M/S 5op7
Redwood Shores, CA 94065
P 650-506-1372
F 650-506-7114

EXAMINATION INDEX

MICHAEL SHAMOS, Ph.D., J.D.
BY MR. PATRAS 7

Certificate of Reporter 329
Errata Sheet 330

EXHIBIT INDEX

PAGE

Shamos
1 Notice of Deposition 7
2 Report of Plaintiff's Expert Michael I. Shamos, Ph.D., J.D., Concerning Invalidity 10
3 Exhibit 3 to Expert Report of Michael Shamos 10
4 Rebuttal Expert Report of Dr. David Finkel 106
5 Expert Report of Dr. David Finkel 126
6 United States Patent No. 5,894,554 133
7 Reference 605, ORCL 00000761-00000781 280

- - - -

PROCEEDINGS

- - - -

THE VIDEOGRAPHER: My name is Adam Werner of Veritext. The date today is June 30th, and the time is approximately 9:10 a.m.

This deposition is being held in the office of Jones Day located at 500 Grant Street, Pittsburgh, PA.

The caption of this case is Oracle Corporation V. epicRealm Licensing in the United States District Court for the District of Delaware.

The name of witness is Michael Shamos.

At this time the attorneys will identify themselves and the parties they represent, after which our court reporter, Heidi Willis of Veritext, will swear in the witness and we can proceed.

MR. PATRAS: Patrick Patras and Benjamin Bradford from Jenner & Block on behalf of epicRealm Licensing. Also on the telephone today is Kevin Meek from Baker Botts also representing epicRealm Licensing.

Page 190

Horizon 601 TCP. So, yeah, 30 was close. Not all of them, but almost.

MR. PATRAS: Okay. Why don't we take a break.

THE WITNESS: Okay.

THE VIDEOGRAPHER: We are now going off the record. The time indicated on the screen is 2:19 p.m.

- - - -

(There was a recess in the proceedings.)

- - - -

THE VIDEOGRAPHER: We are now back on the record. The time indicated on the screen is 2:34 p.m.

BY MR. PATRAS:

Q. Dr. Shamos, are any resources freed on the web server when it receives a TCP acknowledgment?

MR. ARTUZ: Objection, vague, lacks foundation.

A. So the answer is it's not clear with respect to a TCP acknowledgment exactly what happens. The situation at a web server is highly dynamic and fluid. The -- when a TCP session ends, there are certain memory that can be freed to be used for other functions.

Page 191

On the other hand, just before that freeing occurs, there may have been huge gobbling up of memory for some other purpose, so it's not clear that right after the TCP release there's any -- there's any freeing of resource. There could be.

Q. What would have been in that memory that you indicated is released after a TCP acknowledgment is received?

MR. ARTUZ: Same objection.

A. So TCP -- it depends on what you mean by the memory -- so TCP is a protocol. It's defined precisely by internet requests for comment. TCP has to do certain things. It has to allocate buffers to receive data. It has to keep track of the -- there's a numbering scheme for packets in a TCP transfer so that the recipient can determine that he's actually received all of -- all of the packets that constitute the message. That takes space, some amount of space.

As to where this space is, that may differ from implementation to implementation, it may be a fixed amount of memory allocated to the operating system. In other

Page 192

implementations it may be a variable amount of memory allocated with respect to individual processes that are requesting TCP. It could differ.

Anyway, that's what would be contained in memory, information about this particular connection that's open and the data that's being passed back and forth over that connection.

Q. Would there be any processor time that's freed up when the web server receives a TCP acknowledgment?

MR. ARTUZ: Objection, vague, lacks foundation.

A. I'm not sure I know. You know, if there's a table that the processor has to go through and with respect to every open connection regularly, then if you reduce the number of connections by one, then, you know, you would reduce by, you know, a few microseconds the amount of time that the processor would have to spend on that. I don't know if that's actually the case.

Q. Would processor time be used if you had to resend a TCP packet?

Page 193

MR. ARTUZ: Objection, vague, lacks foundation, incomplete hypothetical.

A. Let me try to make it nonvague and complete. So in the TCP protocol is referred to as a reliable protocol, which means that when a message is sent, that if the message is received intact, then an acknowledgment is sent to that effect.

Of course because the internet itself is not reliable, that is not all packets sent are always received, if the sender of a TCP message does not receive an acknowledgment, he doesn't know whether the message was actually received or not, so he tries to resend it, and he keeps resending it until he gets an acknowledgment that it has been received or until some amount of time expires, which is referred to as a time-out.

If there were no time-outs, then the entire internet would be useless at this point because every server in the world would continue trying to send messages that haven't been received for the past six years and there would be no other traffic on the web. So there has to be -- there has to be some kind

Page 194

of time-out.
So, indeed, if an acknowledgment is not received, then assuming that the TCP protocol is used for sending messages, I mean there are other ways of doing it, but if you are using TCP, then the standard way of using TCP is if you don't get the acknowledgement, then after a certain time you attempt to resend, and of course attempting to resend takes some processing time.
Q. When a web server first receives a request for a dynamic web page, is it tasked with responding to that request?
MR. ARTUZ: Same objections.
A. I was following until the word "tasked," because this is client server computing. In client server computing, neither one owns or controls the other. So a client makes requests. A server decides whether it feels like responding to the request or not. It is a voluntary act on the part of the server.
And, now, if the server accepts the responsibility of responding to the request, then you could say it has tasked itself to respond to it in some way. Now, if it's a

Page 195

request for a dynamic page, I suppose the server has a choice of generating the page itself or passing it on to another server for a computationally intensive task, but "tasked" is not a word I would use in that context.
Q. Okay. If a page server did not agree to process the request, would the web server remain responsible for that request?
MR. ARTUZ: Objection, vague, lacks foundation, incomplete hypothetical.
A. So there are any number of systems one could devise. One could devise a system that is able to offer guarantees of a certain kind. One guarantee might be if any page server is capable of processing your request, I will see to it that your request gets processed.
Another form of guarantee might be, hey, I'm going to try to hand off your request to a page server, but, gee, it's not up to me whether that page server is even up or whether it's able to process the request. I gave it -- I gave it a try, and there are various what are referred to as service level options in systems that relate to differing levels of guarantees, and one of those levels is best

Page 196

efforts: You know, I'll try it. If it works, you win. If it doesn't work, I'm not responsible.
And that differs from system -- from system to system, and, of course, the higher degree of guarantee you give, presumably the more you are able to charge for your service or the more desirable your service is.
So it's not true that in every system the web server slavishly keeps trying until the system it succeeds. It depends on how it's built.
Q. Am I correct that by a page server's accepting responsibility of processing the request, the web server is freed of the responsibility of processing that request?
A. Okay. So --
MR. ARTUZ: Same objections.
A. So I'm not buying into the responsibility aspect of this. Responsibility is not a concept that's discussed in the patents. Responsibility is not a concept in client server computing.
As I just said, you could design a system and you could advertise that it offers

Page 197

certain guarantees. For example, you could advertise that my web server takes the responsibility of generating your dynamic page, in which case it would be proper to use the word "responsibility" there. So maybe rephrase in a different way that doesn't use the word "responsibility."
Q. When the client sends a request, does the client have an expectation that the web server will process the request?
MR. ARTUZ: Same objections.
A. Well, there when you are talking about expectations, you are talking about a human emotion. When I sit at someone's website and I type in a URL or some type of request, I have, I'm willing to call it, a hope that it will be responded to, but very many times, you know, you get the message that the website's down or that the server can't connect to the website or that the resource is not available on the website. That's what happens when you have an expired hyperlink. You have an expectation that you are going to get a page, but you don't get the page. So I would characterize it more of a hope than an

Page 198

expectation.
Q. Are those error pages a type of response to the request?
A. Yes. Well, some of them are. 404 not found is a response to the request, but in a 404 not found, the web server actually receives the request, determines that it does not have the resource and sends back a positive acknowledgment with an error code of 404.
If the reason for the nonresponse is that the server is down and it happens by reason of time-out, then no, there is no response from the web server, and it's the sending process that determines that there hasn't been a response, and it sends a message to your browser, so they are different.
Q. If I could direct your attention, Dr. Shamos, to the '554 patent for a moment, and looking at claim 1 of the '554 patent.
A. Yes.
Q. There is a step in the method that reads, Routing said request from said web server to a page server.
A. Yes.
Q. Do you see that?

Page 199

A. Yes.
Q. Does routing in that step necessarily require transmission over a network in your opinion?
MR. ARTUZ: Objection, vague, incomplete hypothetical, lacks foundation.
Q. And I'll limit this question to under Oracle's proposed construction.
A. Let's look over at the proposed construction. Routing says transmitting, that's their -- that's Oracle's proposed construction.
Now, I'm sitting here, I mean I haven't thought about the question, but sitting here now I can envision two ways in which a request might get from a web server to a page server, two fundamentally -- I mean there are many sub ways, but two fundamentally different ways.
If the web server and the page server are on the same physical box, that's a configuration which I do not believe is covered by these patents, but, nonetheless, it's a conceivable configuration.
Then there's interprocess communication that does not involve a network because the machine is not networked to

Page 200

itself, except in a very strange sense. I mean if you want to consider the communications buses that there are on the motherboard to be a network, but normally we wouldn't consider them a network.
Or the page server is on a different physical machine from the web server, in which case the vast majority of conceivable configurations or communications would be over a network.
The way in which it wouldn't would be if there were a single dedicated hardwired line to the page server and there's no real networking going on, but it's effectively a peripheral attached to the web server. I don't know anybody that operates it that way, but it's in the realm of imagination.
Q. Okay. Further in claim 1 there's a reference to routing said request from said web server to a dispatcher. Do you see that?
A. Yes.
Q. Would the routing there necessarily require transmission over a network?
MR. ARTUZ: Objection, lacks foundation, incomplete hypothetical.

Page 201

A. Well, I'll give the same answer. If the dispatcher is a process that's running on the same machine as the web server, then normally we wouldn't consider that to be transmitted over a network. If it's running on a different machine, then it's very probable that it will be a network communication.
Q. Would routing within the same machine meet the claim limitation of routing under Oracle's construction in your view?
MR. ARTUZ: Objection, vague, lacks foundation, incomplete hypothetical.
A. This is a question that cannot be answered in a vacuum. The reason is that the Oracle construction is transmitting, and there are ways in which two processes in a computer can exchange information without transmitting. For example, you can leave data in a given memory location, and the other process picks it up from that memory location. There is no transmission.
On the other hand, there are ways in which it can be done by transmission, by message passing between processes, for example. So the answer is it might be, it

Page 202

might not be, depending on the specific implementation.
Q. You are discussing now, I think, if you want to look at a system whether or not it would meet to that limitation.
My question is whether the limitation, as construed by Oracle, requires there to be necessarily a transmission over a network?
MR. ARTUZ: Same objections.
A. You mean whether transmitting? The Oracle construction of routing is the word "transmitting."
Q. Correct.
A. Are you asking does that transmitting always have to be over a network?
Q. Correct.
A. I don't see why. It could be over a dedicated line, that's not a network. Now, that will get us into a philosophical discussion of what a network is. Are two computers connected by a dedicated line, are they a network or not? And there isn't a sound technical answer to that question. It depends on what you mean by a network.

Page 203

Q. What if you had the web server and dispatcher on a single machine?
A. There may be a transmission. There may not be a transmission. As I said, there are two -- there are different ways of passing information between processes.
Q. If on the single machine scenario there was, in fact, a transmission, would that meet the routing limitation as construed by Oracle in your opinion?
A. Well, I think the --
MR. ARTUZ: Objection, vague, lacks foundation, incomplete hypothetical.
A. I think that you virtually answered your own question. The question is is transmitting transmitting? Yes, transmitting is transmitting. If there is a transmission, there is a transmission. I don't see how that advances us in any way, but it's true.
Q. Would you consider a function call on a single machine between a web server and a dispatcher to meet the routing limitation as construed by Oracle?
MR. ARTUZ: Objection, vague, incomplete hypothetical, lacks foundation.

Page 204

A. It depends on how the request gets to the receiving process. If the request gets to the receiving process by being put in an argument stack or something, then I suppose we could get a bunch of computer scientists in the room and argue over whether that's transmitting or not.
If the method of data communication is that the receiving process goes and it looks in a known memory location for the data, then it's difficult to say that that is transmitting. If I write a note to you on a Post-It note and I leave it here on the table and later on you come around and look at it, have I transmitted something. It's a philosophical question.
Q. Do you believe that it's possible to transmit a request from a web server to a dispatcher on a single machine?
MR. ARTUZ: Objection, incomplete hypothetical, lacks foundation.
A. It is possible.
Q. If I could ask you to turn in your report, Dr. Shamos. Did you go through the routing analysis that you just articulated in

Page 205

determining that the approximately 30 references you believe anticipate the claims perform routing in that manner?
A. I didn't have to because here the one construction subsumes the other.
Q. What do you mean by that?
A. Well, if you -- let's look at epicRealm's construction of routing, if there is one. In the Texas -- I forget now, in the Texas case, was there an agreed construction of routing, or is that one of the terms you didn't require construction of?
In any case I've understood routing to be sending, which is essentially the same thing as transmitting, and so I don't see a difference.
Q. Well, if there's no difference, then I guess the question again becomes did you do the analysis that we just discussed about whether or not there is routing for each of the approximately 30 references that you believe anticipate claim 1 of the '554?
A. There was no need to do a separate analysis because the answer would have been the same in each case.

Page 206

Q. What analysis did you do to determine that there is routing of the request as set forth in claim 1 with respect to the approximately 30 references that you believe anticipate claim 1?
A. I think I just answered that question. I don't consider there to be a substantive difference between epicRealm's and Oracle's construction of routing and, therefore, it's the same analysis. So whatever analysis I did for the one construction applies to the other construction.
Q. Did you determine in doing that analysis whether the request was routed over a network for each of those approximately 30 references that you believe anticipate claim 1 of the '554 patent?
A. I don't think I considered specifically whether it was a network or not.
Q. Okay. Could I direct your attention to paragraph 94 of your report.
A. Yes.
Q. First sentence in paragraph 94 reads, Intercepting is inherent in any system in which the web server processes some but not

Page 207

all requests.
A. Yes.
Q. Do you see that?
A. Yes.
Q. Could you explain what you mean by that?
A. Yes. If the web server is processing all the requests, then there's, of course, no intercepting going on because intercepting is deflecting the request away from the web server so somebody else processes it.
If the web server is processing no requests, then that's a vacuous case that we don't even have to consider.
In the case where the web server is processing some requests and not others, then a determination is being made as to which ones it will process and which ones it isn't. That's what I have taken to be intercepting.
Q. When you refer here to the web server processing the request, could you explain what processing means?
A. Oh, well, this is massively difficult. Okay? Here's the problem. The specification is completely unclear as to what intercepting really is, and it doesn't do a good job of

Page 208

showing it in a drawing. So, in fact, the drawing's inconsistent with the words of the specification.
So the whole issue boils down to what's a web server, and unfortunately neither the parties nor the Court was really precise about what a web server is. It could be two different things, a machine, it could be some software running on a machine, it could be executable, a very specific type running on a machine.
So when we even talk about the web server examining the request, does that mean the web server executable or does that mean other software that might happen to be running on the web server? Totally unspecified in the constructions.
So the real issue is how much work is the web server executable allowed to do on the request before it says no, no, no, this really isn't mine, I abjure from this, I'm passing it off to somebody else.
Under Oracle's construction, it can't do any. The interceptor has to be before the web server executable. The web

Page 209

server executable can't even execute a single instruction, okay?
But that begs the question as to if I show you this web server executable and the first five instructions are the interceptor, you might argue that those first five instructions really aren't part of the web server executable but -- I mean it all depends on where you draw the line, and there isn't enough information in the spec or in the constructions to draw the line in a precise way.
In most situations the interceptor is going to be one line of code. It's going to look for something, it's going to look for a substring in the request, and it's going to say aha, that's a CGI request, I'm sending it off here, in many cases.
And the question is is the web server allowed to do that, or does it have to be something else that we are careful not to call the web server but we call the intercepter that does that.
It's an issue, don't know how to resolve it because there just isn't enough

Page 210

detail in the construction.
Q. Your statement here in paragraph 94 that I read, was that under epicRealm's proposed construction or under Oracle's construction?
A. It's under both. Okay. If the web server doesn't process the request at all, doesn't even look at it, then the intercepting has happened before the web server, and that's Oracle's, that's Oracle's construction.
If the web server processes the request a little bit but it doesn't complete the processing of the request, then that can fall within the definition of the web server not handling the request.
See, the word "handling" wasn't construed. Can you spend three microseconds handling it before you decide you are not handling it anymore? I don't know, and I actually cannot believe that the outcome of this case is going to rest on that -- on that determination.
Q. Under Oracle's proposed construction here, how would the web server receive a request?
A. What's the web server? Is it the machine or the web server executable? You tell me.

Page 211

Q. It's Oracle's construction.
A. No, the language of the claim is epicRealm's language in the claim.
Q. Is it possible under Oracle's proposed construction for the web server to receive a request that is going to be handled ultimately by a page server?
A. Vague and ambiguous. What is the web server? Is it the machine, or is it the web server executable?
Q. You can look at both proposed constructions and answer for each of them.
A. Oh, okay. I can probably do that.
So I'm looking at epicRealm's construction of intercepting, Intercepting said request at said web server, and the epicRealm proposed construction is, intercepting the handling of a request at a web server.
Now, I'm not going to chide people more than usual on this, but you know how it goes with constructions. You saw from my resume that I've written a number of dictionaries. I'm a lexicographer, and I'm really careful about trying to define things,

Page 212

and I really know that any time you try to condense a lot of meaning into a small number of words, you are missing something.
And this construction -- and I'm sure it's also true of Oracle's -- misses something, and intercepting the handling of a request to the web server simply asks the question now what does handling mean. Does handling the request mean doing anything at all to it, or does handling mean completing the processing of the request.
If the handling is simply examining it to decide whether we want to do further handling, then it's doing some handling but it's not doing all the handling.
Oracle wants it so that no handling whatsoever is done by the web server executable. Some other code looks at it first and kicks it away before it gets to the executable.
I can't determine what happens or what's right based on the word "handling." There's just not enough information in the construction to make that determination.
Now let's look at -- now let's look

Page 213

at Oracle's.
Q. Could I ask you a question --
A. Yeah.
Q. -- just based on what you said so far?
A. Sure.
Q. How did you determine that 30 references anticipated the claim if you couldn't tell whether or not something meets intercepting?
A. So I have my own understanding of what intercepted is. I think that it's okay if the web server looks at the request long enough to determine that it shouldn't handle that. I like that. Oracle doesn't like that. I think it's going to be six of one, half dozen of the other.
If Oracle's right, then there's not infringement and maybe the invalidity arguments fall apart. If Oracle's wrong, the invalidity arguments are correct, the patent's invalid, but maybe they would infringe the literal words of the claim. I don't know. It often happens, but it's one or the other. You can't have both.
Q. And do you know how the accused Oracle products work with respect to this issue?

Page 214

A. To --
Q. To the intercepting issue?
A. To save time, I will repeat that I know none of the details of any of the accused instrumentalities. I'm not going to be able to answer that question other than to say no, I don't know. So --
Q. I'm sorry, could I ask one follow up on that?
A. Sure.
Q. Do you know how the Apache web server works with respect to the intercepting issue?
A. I'll say it again. What exactly is the Apache web server? Is it a machine on which Apache happens to be running, or is it the Apache web server executable, and is it the entirety of the executable, or can we delineate certain lines of code that check for dynamic web requests and say that's not really part of the web server, that's part of something else. It all depends on where you draw the line. There isn't enough information to answer the question.
Q. Under what I think you described as your view of intercepting --
A. Yes.

Page 215

Q. -- do you believe that the Apache web server meets that construction, if you will, for intercepting?
A. I believe it does.
Q. Okay. Thank you. I'm sorry, I interrupted your answer.
A. No, no, no, it's your depo. Okay. Now I'm looking at Oracle's construction of intercepting said request at said web server.
Receiving a request at the web server and diverting the request before the web server executable can process the request. That's as I said. They want -- whatever processing you are going to do to determine whether the web server is going to process it, you do it before the web server even gets the teensiest look at one character of the request, okay. That's I think a pretty good lay vision of what intercepting is.
So the quarterback has thrown a pass to you, and I catch it before you do. I have intercepted it. You never even had a chance to try, and then somebody else comes up with the objection that what happens if we both touch it and it's a jump ball and you touched

Page 216

it a little bit but I ended up intercepting it, is that still intercepting. I mean that's what patent cases are about, making arguments like this.
Q. Am I correct that you disagree with Oracle's proposed construction here for intercepting?
A. It's not an issue of agree or disagree. There are sound arguments to be made in favor of either construction. I think, as I said, the lay understanding of interception is the recipient never even touches it. It's been grasped away by somebody else before he gets it. I think that's fine.
Q. But that's not what you personally believe intercepting should be construed to mean; correct?
A. Should be. You know, what I believe is largely irrelevant at this point. The Court's going to make a determination.
Q. But in your opinion you believe that intercepting should be construed differently than what Oracle has proposed here; correct?
A. So when I look at the spec, I have to make sense of a jumble, and the jumble is that it shows the interceptor on completely the wrong

Page 217

end of everything. It shows the interceptor past the web server instead of before the web server.
And I read the words of the spec, and I look at the real world as to how web servers handle CGI requests, and I know that at least based on -- based on the prior art, the web server looks at the request a little bit before it says get the hell out, okay.
And that's the basis on which I think of the two, if you forced, put me against the wall and said you are the judge, which one of these are you going to pick, I'd probably pick the one in which gets to look a little bit, but I see the argument on the other side. And, you know, this is the lexicographer's nightmare. You never come up with the perfect definition of anything.
Q. What construction for intercepting did you use in doing your analysis as to whether the prior art anticipated the claims?
A. I used -- well, it's not clear to me that I used either of them, but I used yours.
Q. Were there any items of prior art that you reviewed, that you recall that would have met

Page 274

foundation.
A. I will have to look. It's the same or similar situation to Oracle 1.0. It's possible that for some entire claims, or maybe even all of them, I never actually have to delve into Reference 605. I don't know that to be the case without looking, but I can look.
Q. Well, I think I understand what you would do if you did look, and so we can go back and look at a different time and not waste the little time we have left together doing that.
A. Oh, I think we'll be seeing each other again, at least somebody else from your side.
Q. Other than the two anticipation arguments that you've identified here with respect to Oracle 2.0, do you have any other anticipation arguments relating to Oracle 2.0?
A. I am not prepared to testify to any such anticipation arguments that are not in my report, understanding that the report includes all the exhibits thereto.
Q. There are no other anticipation arguments you can identify for me besides the two that you identified with respect to Oracle 2.0; is that correct?

Page 275

A. I don't necessarily want to characterize it as two. There may be more than two because within the same row of the spreadsheet, there may be alternative theories within the same row, but there are no others than the ones expressed in the spreadsheet and in the main body of the report and on any other exhibits.
Q. Is that the only places we'd have to look?
A. I think so.
Q. Do you have an opinion as to whether Oracle 2.0 is prior art?
MR. ARTUZ: Objection, vague.
A. So I have taken it to be prior art. I don't -- haven't formed and can't form an independent opinion as to whether it is because dates of conception haven't been established yet, and I don't know the best dates for Oracle 2.0 or 2.0.2. I mean there are some dates listed in manuals, but I don't know.
Q. What is the earliest date that you are aware of that Oracle 2.0 could be effective as a reference?
A. I don't know. I don't know the earliest date that I know, and let me explain that.

Page 276

When I was preparing this spreadsheet, the dates that I put in there for each reference were either the date that was expressly listed in the reference or, in some cases for scientific papers that listed no date, I did an independent science citation search to find what conference those papers were presented at, and I used the date of the conference.
So in this case when I say that Oracle Web Server User's Guide 2.0.2 is March 1996, I believe I got that from the reference itself. Now, it's possible that there may have been deposition testimony that took place after I started preparing this chart that establishes an earlier date. I don't know, maybe.
Q. What's the earliest date for Oracle 2.0 that you are currently aware of?
MR. ARTUZ: Objection, vague, lacks foundation.
A. Okay. Oracle 2.0, the system -- the first release of Oracle 2.0?
Q. Whatever you are relying on for anticipation purposes.

Page 277

A. Well, I'm not --
Q. You mentioned a date of March 1996, for example. Are you aware of any earlier dates than that?
A. So sitting here now without the benefit of being able to look at deposition testimony and only relying on dates that are expressly listed in the reference itself, this second I'm not aware of any date earlier than March 1996. That isn't to say that it doesn't exist, but I'm not being relied upon for establishing dates for prior art references.
Q. Did you do any testing of Oracle 2.0 in your work in this case?
A. No.
Q. Did you examine the source code for Oracle 2.0 in your work in this case?
A. No. The only source code that I may have looked at would be source code quoted in Dr. Finkel's reports. I didn't independently look at them.
Q. Sorry.
A. I did not independently look at any source code.
Q. How were you able to reach your conclusion

that Oracle 2.0 anticipated the asserted claims if you did not examine at the source code?

A. Oh, it's easy. With respect to computer patents, very often it's possible to establish that an entire claim is anticipated because the claim is recited in high level functional terms and the reference itself shows that those functions are performed.

In some other cases where a very specific manner of performing a step is specified in the patent, it may be actually necessary to look at the source code to determine if it's done that way. That was not necessary here because if you look at the claims of the patents, they are in very high level functional terms.

Q. That's true in your opinion of all the asserted claims of the patents-in-suit?

A. Okay. So there is -- let's take an example of the -- of an OLE claim, all right, an OLE claim. If the reference discloses that OLE is used, that's enough. If it doesn't disclose it, then it may be necessary to actually go to the code to see if OLE is used, but very often

you can determine anticipation purely from the documentation of the system itself.

In any case, so we are talking about two different things. One is the reference as an anticipating publication versus the system as prior use. When you are talking about the publication, the publication teaches what it teaches. If you are using the system itself, then you need somebody to come and testify that the documentation, at least with respect to these functions, fairly describes the system that was in operation as of such and such date.

Q. Am I correct that you did not feel it was necessary to do that type of analysis in making your conclusion that Oracle 2.0 anticipated the asserted claims?

A. Okay. When you say that type of analysis, you mean looking at source code?

Q. Correct.

A. That's right.

MR. PATRAS: I'll ask the court reporter to mark as Shamos Exhibit 7 a copy of what is Reference 605, according to the label on the front, and bears Bates numbers Oracle

761 through 781.

- - - -

(Exhibit No. 7 marked for identification.)

- - - -

Q. Dr. Shamos, have you seen Exhibit 7 before?

A. Yes.

Q. What is Exhibit 7?

A. Exhibit 7 is denominated Oracle Web Server 2.0 Technical Note, March 1996.

Q. Do you know whether this document accurately describes the functionality of Oracle 2.0?

A. No, I don't know. I assume it does.

Q. Did you rely on this document for your opinions about the functionality of Oracle 2.0?

A. I'm not sure, as I said before, if we go down the row of the spreadsheet, if I ever make reference to the technical note, then, yes, I did rely on it. But it's not clear that -- I haven't done that exercise, so it's not clear which, if any, claims need the technical note for their support.

I see some references to the technical note in claim 1, for example. Steps 1 -- step 1b6, there's a quote from the

technical note, although that may be simply cumulative because -- yes, so there are a number of quotes. If you look at box, the box corresponding to 1b6 on the row Oracle Web Server 2.0.2, there are a number of quotes from the Web Server User's Guide itself, there's a reference to page A-13. Then later on after that there's another quote from the technical note, which I take to be cumulative evidence. So it may well be that I didn't need to rely on the technical note, but I just threw it in as additional evidence.

Likewise in element 1c, which is processing, the concurrent processing, the first quotation is page 1-2 of the Web Server User's Guide, but the last quotation is from the technical note, and so it's cumulative.

And so we'd have to go through a collation to determine exactly which claims require the technically note for anticipation and which ones don't. It appears to me just looking at this that claim 1 does not require it.

Q. Could I ask you to turn in Exhibit 7, Dr. Shamos, to the page with Bates No. ending

Exhibit 10

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 11

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 12

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 13

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC., **Plaintiffs,** **v.** **EPICREALM LICENSING, LLC,** **Defendant.**	**C.A. No. 06-cv-414 (SLR)**
AND RELATED COUNTERCLAIMS	

REPORT OF PLAINTIFFS' EXPERT MICHAEL I. SHAMOS, PH.D, J.D. CONCERNING INVALIDITY

BACKGROUND & QUALIFICATIONS

1. My name is Michael I. Shamos. I hold the title of Distinguished Career Professor in the School of Computer Science at Carnegie Mellon University in Pittsburgh, Pennsylvania. I was a founder and Co-Director of the Institute for eCommerce at Carnegie Mellon and I now direct a graduate degree program in eBusiness Technologies. My résumé is attached as Exhibit 2 to this report.

2. I currently teach graduate courses at Carnegie Mellon in Electronic Commerce, including eCommerce Technology, Electronic Payment Systems, Electronic Voting and eCommerce Law and Regulation and have done so since 1999. In Fall 2007 I taught a new course entitled Law of Computer Technology.

every claim is anticipated under one or more of epicRealm's varieties of "releasing." Every one of the references in Exhibit 3 utilizes implicit releasing in addition to any releasing it may perform under epicRealm's contentions.

e. "intercepting"

92. The Texas Court has construed "intercepting said request at said Web server" to mean "intercepting the handling of a request at a Web server."[5] It is my understanding that epicRealm has proposed the same construction for 'intercepting' in this action. This construction clarifies that intercepting involves not only redirecting the request itself, but also the processing (handling) of the request. My analysis below is also consistent with Oracle's proposed construction for the term "intercepting," namely, "receiving a request at the Web server and diverting the request before the Web server executable can process the request."

93. Intercepting is important to maintain server throughput. If a web server must spend any significant time analyzing a request before determining that it cannot handle the request, its efficiency will be diminished. It is therefore desirable to intercept such requests, ideally before they are even processed by the server.

94. Intercepting is inherent in any system in which the Web server processes some but not all, requests. Some mechanism must be used to discriminate between the requests the Web server will handle and which it will not. That mechanism which prevents the Web server from handling certain requests "intercepts" their handling within the Texas Court's construction.

95. Intercepting the handling of requests was well known in the prior art, as discussed in connection with Eckerson, above. Intercepting CGI requests (used to generate dynamic web pages) is taught in reference 166, Bittinger et al. U.S. Patent 5,754,774: "This web server originated communication is intercepted by the server-side intercept module 40 and transformed by a client/server specific data stream." (10:12-14). "After receiving the information from the client-side intercept module 30, the server-side intercept module 40 checks its server cache resident in the second computer to determine if a server cache entry exists corresponding to the

[5] There is a parallel construction for "intercepting said request at said HTTP-compliant device."

Exhibit 14

Oracle9*i* Application Server

Performance Guide

Release 2 (9.0.2)

April 2002

Part No. A95102-02

ORACLE

Oracle9*i* Application Server Performance Guide, Release 2 (9.0.2)

Part No. A95102-02

Copyright © 2002 Oracle Corporation. All rights reserved.

Primary Author: Thomas Van Raalte

Contributors: Eric Belden, Paul Benninghoff, Alice Chan, Greg Cook, Marcelo Goncalves, Helen Grembowicz, Bruce Irvin, Pushkar Kapasi, Paul Lane, Sharon Malek, Valarie Moore, Carol Orange, Julia Pond, Leela Rao, Joan Silverman, Sanjay Singh, Cheryl Smith, Zhunquin Wang, Brian Wright

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle*MetaLink*, Oracle Store, Oracle9i, Oracle9iAS Discoverer, SQL*Plus, and PL/SQL are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

6

Optimizing J2EE Applications In OC4J

This chapter provides guidelines for improving the performance of Oracle9*i*AS Containers for J2EE (OC4J) applications in Oracle9*i* Application Server.

This chapter contains:

- OC4J J2EE Application Performance Quickstart
- Improving J2EE Application Performance by Configuring OC4J Instance
- Improving Servlet Performance in Oracle9iAS
- Improving JSP Performance in Oracle9iAS
- Improving EJB Performance in Oracle9iAS
- Using Multiple OC4Js and Limiting Connections
- Database Monitoring and Tuning
- Improving BC4J Performance in Oracle9iAS

Note: This chapter describes using Oracle Enterprise Manager for setting OC4J and application configuration options. You can also use the Distributed Configuration Management (DCM) utility, `dcmctl`, to set configuration options. This utility provides a command-line alternative to using Oracle Enterprise Manager for some Oracle9*i*AS configuration and management tasks.

OC4J J2EE Application Performance Quickstart

This section provides a quickstart for tuning J2EE applications that run on OC4J, providing links for information on important performance issues.

Table 6–1 lists a quick guide for performance issues for J2EE applications.

Table 6–1 Critical Performance Areas for J2EE Applications

Performance Area	Description and Reference
Providing Adequate Memory Resources	To improve the performance of your J2EE applications, provide adequate memory resources. If the OC4J running your J2EE applications does not have enough memory, performance can suffer due to the overhead required to manage limited memory See "Setting the JVM Heap Size for OC4J Processes" on page 6-3
Caching and Reusing Database Connections	Setting up database connection pooling properly is often a critical performance consideration for J2EE applications that access a database. Data sources provide configuration options that allow you to use and configure pooled database connections. See "Setting Up Data Sources – Performance Issues" on page 6-9
Managing Concurrency and Limiting Connections	See "Limiting HTTP Connections" on page 6-35
Load Balancing	See "Configuring Multiple OC4J Processes" on page 6-37
Balancing Applications	See "Balancing Applications Across OC4J Instances" on page 6-38
Database Monitoring and Tuning	See "Database Monitoring and Tuning" on page 6-38

Improving J2EE Application Performance by Configuring OC4J Instance

Tuning OC4J configuration options allows you to improve the performance of J2EE applications running on an OC4J Instance. Modifying the configuration may require balancing the available resources on your system with the performance requirements for your applications.

This section covers configuration changes that can affect J2EE application performance and includes the following topics:

- Setting Java Options for OC4J Processes
- Setting Up Data Sources – Performance Issues

See Also:

Chapter 3, "Monitoring Oracle HTTP Server"

Chapter 4, "Monitoring OC4J"

Setting Java Options for OC4J Processes

When running Oracle9*i*AS, the module `mod_oc4j` is the connector from Oracle HTTP Server to one or more OC4J Instances. Each OC4J process within an OC4J Instance runs in its own Java Virtual Machine (JVM) and is responsible for parsing J2EE requests and generating a response. When a request comes into Oracle HTTP Server, `mod_oc4j` picks an OC4J process and routes the request to the selected OC4J process. Within each OC4J Instance all of the OC4J JVM processes use the same configuration and start with the same Java options. Likewise, unless a process dies or there is some other problem, each OC4J process that is part of an OC4J Instance has the same J2EE applications deployed to it.

Depending on your J2EE application, you may be able to improve the application's performance by setting Java Options for the JVM running OC4J where your application is deployed.

Setting the JVM Heap Size for OC4J Processes

If you have sufficient memory available on your system and your application is memory intensive, you can improve your application performance by increasing the JVM heap size from the default values. While the amount of heap size required varies based on the application and on the amount of memory available, for most

OC4J server applications, a heap size of at least 128 Megabytes is advised. If you have sufficient memory, using a heap size of 256 Megabytes or larger is preferable.

To change the size of the heap allocated to the OC4J processes in an OC4J Instance, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-7, and specify the following Java options:

```
-Xmssizem -Xmxsizem
```

Where *size* is the desired Java heap size in megabytes.

If you know that your application will consistently require a larger amount of heap, you can improve performance by setting the minimum heap size equal to the maximum heap size, by setting the JVM `-Xms` size to be the same as the `-Xmx` size.

For example, to specify a heap size of 128 megabytes, specify the following:

```
-Xms128m -Xmx128m
```

You should set your maximum Java heap size so that the total memory consumed by all of the JVMs running on the system does not exceed the memory capacity of your system. If you select a value for the Java heap size that is too large for your hardware configuration, one or more of the OC4J processes within the OC4J Instance may not start, and Oracle Enterprise Manager reports an error. Review the log files for the OC4J Instance in the directory `$ORACLE_HOME/opmn/logs`, to find the error report:

```
Could not reserve enough space for object heap
Error occurred during initialization of VM
```

If you select a value for the JVM heap size that is too small, none of the OC4J processes will be able to start, and Oracle Enterprise Manager reports an error. If you review the log files for the OC4J Instance in the directory `$ORACLE_HOME/opmn/logs`, you may find errors similar to the following:

```
java.lang.OutOfMemoryError
```

Note: There are other reasons why `java.lang.OutOfMemoryError` error may occur. For example, if the application has a memory leak.

If the system runs out of memory, the OC4J process will shut down. This will happen if references to the objects are not released. For example, if objects are stored in a hash table or vector and never again removed.

It is of course possible that your process actually needs to use a lot of memory. In this case, the maximum heap size for the process should be increased to avoid frequent garbage collection.

To maximize performance, set the maximum heap size to accommodate application requirements. To determine how much Java heap you need, include calls in your program to the `Runtime.getRuntime().totalMemory()` and `Runtime.getRuntime().freeMemory` methods in the `java.lang` package. Subtract free memory from total memory; the difference is the amount of heap that the application consumed.

> **See Also:** You can find detailed information about JVM options and their impact on performance on the JVM vendor's web sites.

Setting the Server Option for OC4J Processes (for UNIX)

Depending on the particular J2EE application, setting the command line option `-server` for the JVM running OC4J may improve performance (the JVM runs in one of two modes set with the two related options, `-client` and `-server`, the default value is `-client`). To set this option, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-7, and specify the `-server` Java option.

> **Note:** Do not use the `-server` option when running OC4J on Oracle9*i* Application Server for Windows systems. Our tests indicate that using the `-server` option on JVMs running OC4J on Windows systems does not improve and may degrade performance.

The `-server` option selects the server VM instead of using the default client VM. The client and the server VMs are similar, except that the server VM is specially tuned to maximize peak operating speed. It is intended for executing long-running server applications, for which having the fastest possible operating speed is generally more important than having a fast startup time or a smaller runtime memory footprint. The client VM, the default without using the `-server` option starts up faster and requires a smaller memory footprint than the server VM.

> **Note:** The `-server` option must be specified first, before all other Java options.

Setting the Stack Size Option for OC4J Processes

Depending on the particular J2EE application, changing the setting of the command line option `-Xss` for the JVM running OC4J may improve performance. To set this option, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-7, and specify the `-Xss` Java option.

This option sets the maximum stack size for C code in a thread to *n*. Every thread that is spawned during the execution of the program passed to java has *n* as its C code stack size. The default C code stack size is 512 kilobytes (`-Xss512k`). A value of 64 kilobytes is the smallest amount of C code stack space allowed per thread.

Oracle recommends that you try the following value to improve the performance of your J2EE applications:

```
-Xss128k
```

Setting the Concurrentio Option for OC4J Processes

Depending on the particular J2EE application and JDK version, changing the setting of the command line option `-Xconcurrentio` for the JVM running OC4J may improve performance. To set this option, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-7, and specify the `-Xconcurrentio` Java option.

This option generally helps programs with many threads. The main feature turned on with `-Xconcurrentio` is to use LWP based synchronization instead of thread based synchronization. For certain applications, with JDK 1.3.1, this option increases speed up by over 40%. See the following site for more information,

```
http://java.sun.com/docs/hotspot/threads/threads.html
```

Using the `-Xconcurrentio` option, it is important to compare results for your application without the option. In some tests, results have been mixed, using the option results in a speedup for some applications, but for other applications or JDK versions, the performance degraded with this option. See the following site,

```
http://java.sun.com/docs/hotspot/PerformanceFAQ.html
```

Note: In JDK 1.4, LWP based synchronization is the default, but setting `-Xconcurrentio` can still improve performance in JDK 1.4 since it turns on additional internal options.

Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options

To change the Java command line options for an OC4J Instance, go to the home page for the OC4J Instance and perform the following steps:

1. Stop the OC4J Instance.
2. Drill down to the Server Properties page.
3. In the Command Line Options area of the Server Properties page, under the heading Multiple VM Configuration, set the Java Options.

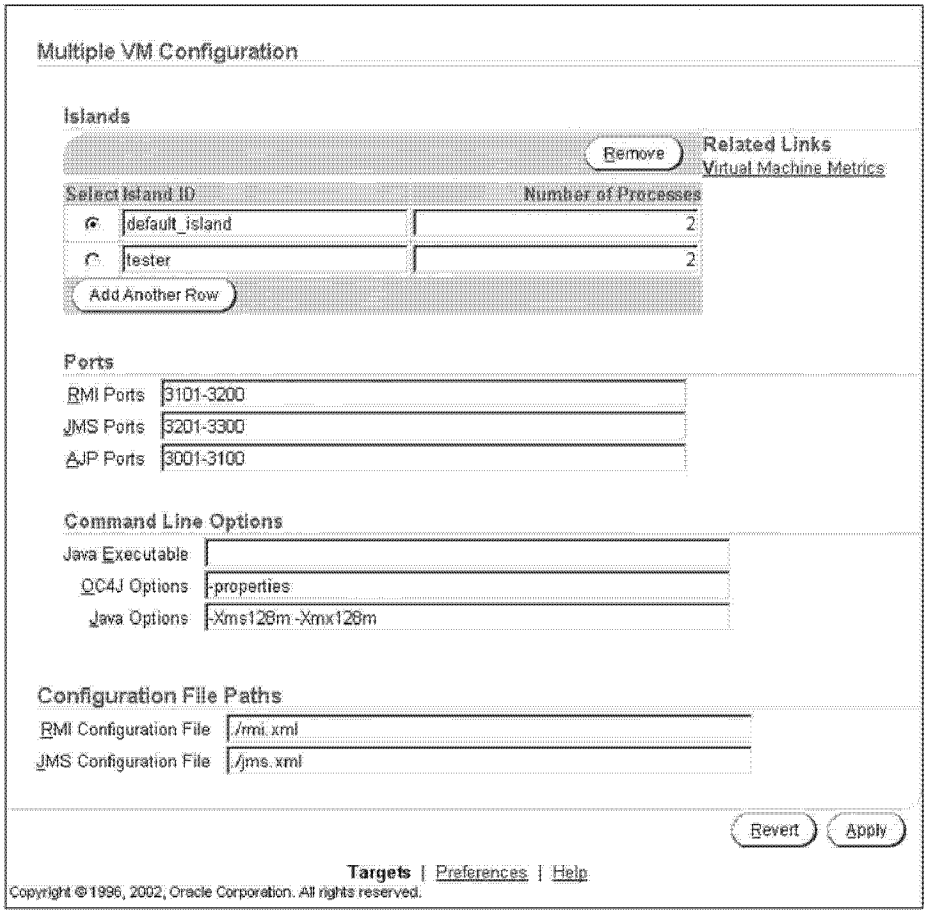
   For example, enter the following to set the JVM heap sizes to 128 Megabytes:

   ```
   -Xmx128m
   ```

4. Use the Apply button to apply the changes.
5. Start the OC4J Instance.

Figure 6–1 shows the Server Properties page with Java Options.

Figure 6–1 Setting Java Heap Size for an OC4J Instance Using Oracle Enterprise Manager



Setting Up Data Sources – Performance Issues

A data source, which is the instantiation of an object that implements the `javax.sql.DataSource` interface, enables you to retrieve a connection to a database server. This section describes data source configuration options for global data sources. A global data source is available to all the deployed applications in an OC4J Instance.

This section covers the following topics:

- Emulated and Non-Emulated Data Sources
- Using the EJB Aware Location Specified in Emulated Data Sources
- Setting the Maximum Open Connections in Data Sources
- Setting the Minimum Open Connections in Data Sources
- Setting the Cached Connection Inactivity Timeout in Data Sources
- Setting the Wait for Free Connection Timeout in Data Sources
- Setting the Connection Retry Interval in Data Sources
- Setting the Maximum Number of Connection Attempts in Data Sources
- Using Oracle Enterprise Manager to Change Data Source Configuration Options

---

Note: If your data source is provided by a third party, you may need to set certain properties. These properties should be defined in the third-party documentation.

See Also:

- "Improving EJB Performance in Oracle9iAS" on page 6-28
- Chapter 4, "Data Sources Primer" in *Oracle9iAS Containers for J2EE User's Guide*
- Chapter 15, "Data Sources" in *Oracle9iAS Containers for J2EE Services Guide*
- *Oracle9iAS Containers for J2EE Enterprise JavaBeans Developer's Guide and Reference*

Emulated and Non-Emulated Data Sources

Some of the performance related configuration options have different affects, depending on the type of the data source. OC4J supports two types of data sources, emulated and non-emulated:

The pre-installed default data source is an emulated data source. Emulated data sources are wrappers around Oracle data sources. If you use these data sources, your connections are extremely fast, because they do not provide full XA or JTA global transactional support. We recommend that you use these data sources for local transactions or when your application requires access or update to a single database. You can use emulated data sources for Oracle or non-Oracle databases.

You can use the emulated data source to obtain connections to different databases by changing the values of the `url` and `connection-driver` parameters.

The following is a definition of an emulated data source:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:5521:oracle"
  inactivity-timeout="30"
/>
```

Non-emulated data sources are pure Oracle data sources. These are used by applications that want to coordinate access to multiple sessions within the same database or to multiple databases within a global transaction.

Using the EJB Aware Location Specified in Emulated Data Sources

Each data source is configured with one or more logical names that allow you to identify the data source within J2EE applications. The `ejb-location` is the logical name of an EJB data source. In addition, use the `ejb-location` name to identify data sources for most J2EE applications, where possible, even when not using EJBs. The `ejb-location` only applies to emulated data sources. You can use this option for single phase commit transactions or emulated data sources.

Using the `ejb-location`, the data source manages opening a pool of connections, and manages the pool. Opening a connection to a database is a time-consuming process that can sometimes take longer than the operation of getting the data itself. Connection pooling allows client requests to have faster response times, because the applications do not need to wait for database connections to be created. Instead, the applications can reuse connections that are available in the connection pool.

Note: Oracle recommends that you only use the `ejb-location` JNDI name in emulated data source definitions for retrieving the data source. For non-emulated data sources, you must use the `location` JNDI name.

Setting the Maximum Open Connections in Data Sources

The `max-connections` option specifies the maximum number of open connections for a pooled data source. To improve system performance, the value you specify for the number `max-connections` depends on a combination of factors including the size and configuration of your database server, and the type of SQL operations that your application performs.

The default value for `max-connections` and the handling of the maximum depends on the data source type, emulated or non-emulated.

For emulated data sources, there is no default value for `max-connections`, but the database configuration limits that affect the number of connections apply. When the maximum number of connections, as specified with `max-connections`, are all active, new requests must wait for a connection to be become available. The maximum time to wait is specified with `wait-timeout`.

For non-emulated data sources, there is a property, `cacheScheme`, that determines how max-connections is interpreted. Table 6–2 lists the values for the `cacheScheme` property (`DYNAMIC_SCHEME` is the default value for `cacheScheme`).

See Also:

- "Setting the Wait for Free Connection Timeout in Data Sources" on page 6-14
- Chapter 15, "Data Sources" in *Oracle9iAS Containers for J2EE Services Guide*

Table 6–2 Non-emulated Data Source cacheScheme Values

Value	Description
`FIXED_WAIT_SCHEME`	In this scheme, when the maximum limit is reached, a request for a new connection waits until another client releases a connection.
`FIXED_RETURN_NULL_SCHEME`	In this scheme, the maximum limit cannot be exceeded. Requests for connections when the maximum has already been reached return null.
`DYNAMIC_SCHEME`	In this scheme, you can create new pooled connections above and beyond the maximum limit, but each one is automatically closed and freed as soon as the logical connection instance is finished being used, where it is returned to the available cache. `DYNAMIC_SCHEME` is the default value for `cacheScheme`.

The tradeoffs for changing the value of `max-connections` are:

- For some applications you can improve performance by limiting the number of connections to the database (this causes the system to queue requests in the mid-tier). For example, for one application that performed a combination of updates and complex parallel queries into the same database table, performance was improved by over 35% by reducing the maximum number of open connections to the database by limiting the value of `max-connections`.

> **Note:** You should check to make sure that your database is configured to allow at least the total number of open connections, as specified by the data sources `max-connections` option for all your J2EE applications.

Setting the Minimum Open Connections in Data Sources

The `min-connections` option specifies the minimum number of open connections for a pooled data source.

For applications that use a database, performance can improve when the data source manages opening a pool of connections, and manages the pool. This can improve performance because incoming requests don't need to wait for a database connection to be established; they can be given a connection from one of the available connections, and this avoids the cost of closing and then reopening connections.

By default, the value of `min-connections` is set to 0. When using connection pooling to maintain connections in the pool, specify a value for `min-connections` other than 0.

For emulated and non-emulated data sources, the `min-connections` option is treated differently.

For emulated data sources, when starting up the initial `min-connections` connections, connections are opened as they are needed and once the `min-connections` number of connections is established, this number is maintained.

For non-emulated data sources, after the first access to the data source, OC4J then starts the `min-connections` number of connections and maintains this number of connections.

Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check to make sure that your database is configured to allow at least as large a number of open connections as the total of the values specified for all the data sources `min-connections` options, as specified in all the applications that access the database.

Note: If the `min-connections` is set to a value other than zero, the specified number of connections will be maintained; OC4J maintains the connections when they are not in use, and they do not time out when the specified `inactivity-timeout` is reached.

Once the specified connections are opened, OC4J does not provide a way to close the connections, except by stopping OC4J.

Setting the Cached Connection Inactivity Timeout in Data Sources

The `inactivity-timeout` specifies the time, in seconds, to cache unused connections before closing them.

To improve performance, you can set the `inactivity-timeout` to a value that allows the data source to avoid dropping and then re-acquiring connections while your J2EE application is running.

The default value for the `inactivity-timeout` is 60 seconds, which is typically too low for applications that are frequently accessed, where there may be some inactivity between requests. For most applications, to improve performance, we recommend that you increase the `inactivity-timeout` to 120 seconds.

To determine if the default `inactivity-timeout` is too low, monitor your system. If you see that the number of database connections grows and then shrinks during an idle period, and grows again soon after that, you have two options: you can increase the `inactivity-timeout`, or you can increase the `min-connections`.

See Also:

- "Setting the Minimum Open Connections in Data Sources" on page 6-12

Setting the Wait for Free Connection Timeout in Data Sources

The `wait-timeout` specifies the number of seconds to wait for a free connection if the connection pool does not contain any available connections (that is, the number of connections has reached the limit specified with `max-connections` and they are all currently in use).

If you see connection timeout errors in your application, increasing the `wait-timeout` can prevent the errors. The default `wait-timeout` is 60 seconds.

If database resources, including memory and CPU are available and the number of open database connections is approaching `max-connections`, you may have limited `max-connections` too stringently. Try increasing `max-connections` and monitor the impact on performance. If there are not additional machine resources available, increasing `max-connections` is not likely to improve performance.

You have several options in the case of a saturated system:

- Increase the allowable `wait-timeout`.
- Evaluate the application design for potential performance improvements.
- Increase the system resources available and then adjust these parameters.

Setting the Connection Retry Interval in Data Sources

The `connection-retry-interval` specifies the number of seconds to wait before retrying a connection when a connection attempt fails.

If the `connection-retry-interval` is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for the `connection-retry-interval` is 1 second.

Setting the Maximum Number of Connection Attempts in Data Sources

The `max-connect-attempts` option specifies the maximum number of times to retry making a connection. This option is useful to control when the network is not stable, or the environment is unstable for any reason that sometimes makes connection attempts fail.

If the `connection-retry-interval` option is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for `max-connect-attempts` is 3.

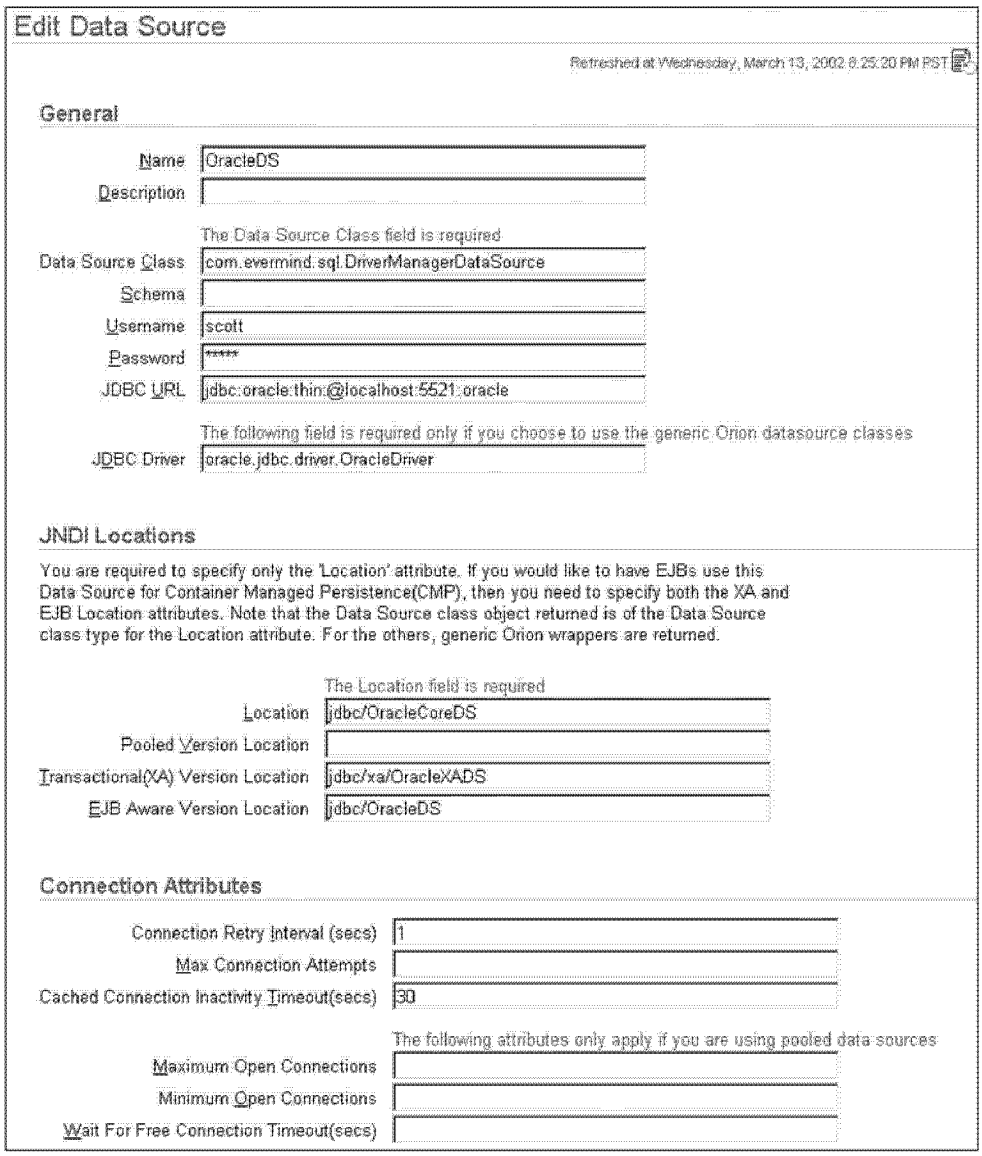
Using Oracle Enterprise Manager to Change Data Source Configuration Options

Figure 6–2 shows the Oracle Enterprise Manager configuration page that lets you view or modify a data source. This page is available in Oracle Enterprise Manager by selecting the Edit button for a selected data source from the Data Sources page from the application default page for an OC4J Instance, or by selecting data sources from the administration section of a deployed application's description page (this is only available when the application has its own local data source).

Oracle Enterprise Manager stores the data sources elements that you add or modify in an XML file. This file defaults to the name `data-sources.xml` and is located in `/j2ee/home/config`. If you want to change the name or the location of this file, you can do this in the General Properties page off of the default application screen or off of your specific application's page, when the application specifies a local data source.

Note: You can also use the Oracle Enterprise Manager Advanced Properties links to create or edit data sources. This allows you to add data sources using the XML definitions which is useful if you have been provided the XML.

Figure 6–2 Oracle Enterprise Manager Data Sources Configuration Page



Improving Servlet Performance in Oracle9*i*AS

This section discusses configuration options and performance tips specific to servlets for optimizing OC4J performance.

This section covers the following topics:

- Improving Performance by Altering Servlet Configuration Parameters
- Servlet Performance Tips

Improving Performance by Altering Servlet Configuration Parameters

This section cover the following:

- Loading Servlet Classes at Startup

Loading Servlet Classes at Startup

By default, OC4J loads a servlet when the first request is made. OC4J also allows you to load servlet classes when the JVM that runs the servlet is started. To do this, add the `<load-on-startup>` sub-element to the `<servlet>` element in the application's `web.xml` configuration file.

For example, add the `<load-on-startup>` as follows:

```
<servlet>
      <servlet-name>viewsrc</servlet-name>
      <servlet-class>ViewSrc</servlet-class>
      <load-on-startup>
</servlet>
```

Using the load-on-startup facility increases the start-up time for your OC4J process, but decreases first-request **latency** for servlets.

Using Oracle Enterprise Manager you can also specify that OC4J load for an entire Web Module on startup. To specify that a web module is to be loaded on startup, select the Website Properties page for an OC4J Instance and then use the Load on Startup checkbox.

Servlet Performance Tips

The following tips can enable you to avoid or debug potential performance problems:

- Analyze Servlet Duration
- Understand Server Request Load
- Find Large Servlets That Require a Long Load Time
- Watch for Unused Sessions
- Watch for Abnormal Session Usage
- Load Servlet Session Security Routines at Startup

Analyze Servlet Duration

It is useful to know the average duration of the servlet (and JSP) requests in your J2EE enterprise application. By understanding how long a servlet takes when the system is not under load, you can more easily determine the cause of a performance problem when the system is loaded. The average duration of a given servlet is reported in the metric `service.avg` for that servlet. You should only examine this value after making many calls to the servlet so that any startup overhead such as class loading and database connection establishment will be amortized.

As an example, suppose you have a servlet for which you notice the `service.avg` is 32 milliseconds. And suppose you notice a response time increase when your system is loaded, but not CPU bound. When you examine the value of `service.avg`, you might find that the value is close to 32 ms, in which case you can assume the degradation is probably due to your system or application server configuration rather than in your application. If on the other hand, you notice that `service.avg` has increased significantly, you may look for the problem in your application. For example, multiple users of the application may be contending for the same resources, including but not limited to database connections.

> **See Also:** Table A–11 in Appendix A, "Oracle9iAS Performance Metrics"

Understand Server Request Load

In debugging servlet and JSP problems, it is often useful to know how many requests your OC4J processes are servicing. If the problems are performance related, it is always helpful to know if they are aggravated by a high request load. You can

track the requests for a particular OC4J Instance using Oracle Enterprise Manager, or by viewing the application's web module metrics.

> **See Also:** Table A–11 in Appendix A, "Oracle9iAS Performance Metrics"

Find Large Servlets That Require a Long Load Time

You may find that a servlet application is especially slow the first time it is used after the server is started, or that it is intermittently slow. It is possible that when this happens the server is heavily loaded, and the response time is suffering as a result. If there is no indication of a high load, however, which you can detect by monitoring your access logs, periodically monitoring CPU utilization, or by tracking the number of users that have active requests on the HTTP server and OC4J, then you may just have a large servlet that takes a long time to load.

You can see if you have a slow loading servlet by looking at `service.maxTime`, `service.minTime`, and `service.avg`. If the time to load the servlet is much higher than the time to service, the first user that accesses the servlet after your system is started will feel the impact, and `service.maxTime` will be large. You can avoid this by configuring the system to initialize your servlet when it is started.

> **See Also:** "Loading Servlet Classes at Startup" on page 6-17

Watch for Unused Sessions

You should regularly monitor your applications looking for unused sessions. It is easy to inadvertently write servlets that do not invalidate their sessions. Without source code for the application software, you may not know this could be a problem on your host, but sooner or later you would notice a higher consumption of memory than expected. You can see if there are sessions which are not utilized or sessions which are not being properly invalidated after being used with the session metrics, including: `sessionActivation.time` and `sessionActivation.completed` and `sessionActivation.active`.

> **See Also:** Table A–12 in Appendix A, "Oracle9iAS Performance Metrics"

Watch for Abnormal Session Usage

This example shows an application that creates sessions, but never uses them:

To provide an example, we show metrics from a JSP under `/oc4j/application/WEBs/context`:

```
session.Activation.active:       500 ops
session.Activation.completed:      0 ops
```

This application created 500 sessions and all are still active. Possibly, this indicates that the application makes unnecessary use of the sessions and it is just a matter of time before it causes memory or CPU consumption problems.

A well-tuned application shows `sessionActivation.active` with a value that is less than `sessionActivation.completed` before the session time out. This indicates that the sessions are probably being used and cleaned up.

Suppose we have a servlet that uses sessions effectively and invalidates them appropriately. Then we might see a set of metrics such as the following, under `/oc4j/<application>/WEBs/<context>`:

```
session.Activation.active:         2 ops
session.Activation.completed:    500 ops
```

The fact that two sessions are active when more than 500 have been created and completed indicates that sessions are being invalidated after use.

Load Servlet Session Security Routines at Startup

OC4J uses the class `java.security.SecureRandom` for secure seed generation. The very first call to this method is time consuming. Depending on how your system is configured for security, this method may not be called until the very first request for a session-based servlet is received by the Application Server. One alternative is to configure the application to load-on-startup in the application's `web.xml` configuration file and to create an instance of `SecureRandom` during the class initialization of the application. The result will be a longer startup time in lieu of a delay in servicing the first request.

See Also: "Loading Servlet Classes at Startup" on page 6-17

Improving JSP Performance in Oracle9*i*AS

OracleJSP is Oracle's implementation of the Sun Microsystems JavaServer Pages specification. Some of the additional features it includes are custom JavaBeans for accessing Oracle databases, SQL support, and extended datatypes.

This section explains how you can improve OracleJSP performance. It contains the following topics:

- Improving Performance by Altering JSP Configuration Parameters
- Improving Performance by Tuning JSP Code

> **Note:** A JSP is translated into a Java servlet before it runs, therefore servlet performance issues also apply for JSPs.

Oracle9*i*AS provides JSP tag libraries that include some features that may improve the performance of J2EE applications. For example, you may be able to use the JSP caching features available in the tag libraries to increase the speed and scalability for your applications:

- The JESI tag library supports the use of Oracle9*i*AS Web Cache. This supports the use of the HTTP-level cache, maintained outside the application, that provides very fast cache operations. Oracle9*i*AS Web Cache is capable of caching static data, such HTML, GIF, or JPEG files, or dynamic data, such as servlet or JSP results.
- The Web Object Cache tag library allows you to capture intermediate results of JSP and servlet execution, and subsequently reuse these cached results in other parts of the Java application logic.

See Also:

- "Improving JSP Performance in Oracle9iAS" on page 6-21
- Chapter 5, "Key Considerations" in *Oracle9iAS Containers for J2EE Servlet Developer's Guide*
- *Oracle9iAS Containers for J2EE JSP Tag Libraries and Utilities Reference*

Improving Performance by Altering JSP Configuration Parameters

This section describes JSP configuration parameters that you can alter to improve and control JSP operation. These parameters are set for each OC4J Instance, by altering the file `global-web-application.xml`.

See Also:

- *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* in the Oracle9*i* Application Server documentation library for information on JSP configuration parameters.
- *Oracle9iAS Containers for J2EE Servlet Developer's Guide* in the Oracle9*i* Application Server documentation library for information on `global-web-application.xml`

Using the main_mode Parameter

The `main_mode` parameter determines whether classes are automatically reloaded or JSPs are automatically recompiled, in case of changes.

Table 6–3 shows the supported settings for `main_mode`.

Table 6–3 JSP main_mode Parameter Values

Option	Description
`justrun`	The runtime dispatcher does not perform any timestamp checking, so there is no recompilation of JSPs or reloading of Java classes. This mode is the most efficient mode for a deployment environment, where code will not change. If comparing timestamps is unnecessary, as is the case in a typical production deployment environment where source code will not change, you can avoid all timestamp comparisons and any possible retranslations and reloads by setting the `main_mode` parameter to the value `justrun`. Using this value can improve the performance of JSP applications. Note: before you set `main_mode` to the value `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a browser, or by running your application (using the default value for `main_mode`, `recompile`). This assures that the JSP is compiled before you set the `justrun` flag.
`reload`	The dispatcher will check if any classes have been modified since loading, including translated JSPs. JavaBeans invoked from pages, and any other dependency classes.
`recompile`	This is the default value for `main_mode`. The dispatcher will check the timestamp of the JSP, retranslate it if it has been modified since loading, and execute all `reload` functionality as well.

Note the following when working with the `main_mode` parameter:

- Because of the usage of in-memory values for class file last-modified times, removing a page implementation class file from the file system will ***not*** by itself cause retranslation of the associated JSP source.
- The page implementation class file will be regenerated when the memory cache is lost. This happens whenever a request is directed to this page after the server is restarted or after another page in this application has been retranslated.
- A page is ***not*** reloaded just because a statically included file has changed. Statically included files, included through `<%@ include ... %>` syntax as opposed to `<jsp:include ... />` syntax, are included during translation-time.

> **Note:** Before you set `main_mode` to the value `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a browser, or by running your application.

Improving Performance by Tuning JSP Code

This section describes changes you can make to your JSP code to improve performance.

This section covers the following topics:

- Impact of Session Management on Performance
- Using Static Template Text Instead of out.print for Outputting Text
- Performance Issues for Buffering JSPs
- Using Static Versus Dynamic Includes

Impact of Session Management on Performance

In general, sessions add performance overhead for your Web applications. Each session is an instance of the `javax.servlet.http.HttpSession` class. The amount of memory per session depends on the size of the session objects created. You can turn off sessions for your JSPs if you do not want a new session created for each request. By default, in OracleJSP sessions are enabled. If you do not need to use sessions in your JSPs, turn them off by including the following line at the top of the JSP:

```
<%@ page session="false" %>
```

If you use sessions, ensure that you explicitly cancel the session. If you do not cancel a session, it remains active until it times out. Invoke the `invalidate()` method to cancel a session.

The default session timeout for OC4J is 30 minutes. You can change this for a specific application by setting the `<session-timeout>` parameter in the `<session-config>` element of `web.xml`.

For example, the following code shows how you would cancel a session after you have finished using it:

```
HtttpSession session;
session = httpRequest.getSession(true);
.
.
.
session.invalidate();
```

OC4J uses the class `java.security.SecureRandom` for secure seed generation. The very first call to this method is time consuming. Depending on how your system is configured for security, this method may not be called until the very first request for a session-based JSP is received by the Application Server. One alternative is to force this call to be made on startup by including a call in the class initialization for some application that is loaded on startup. The result will be a longer startup time in lieu of a delay in servicing the first request.

Note: JSP pages by default use sessions while servlets by default do not use sessions.

See Also:

- *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* for information on sessions
- *Oracle9iAS Containers for J2EE Servlet Developer's Guide* for information on sessions

Using Static Template Text Instead of out.print for Outputting Text

Using the JSP code `out.print("<html>")` requires more resources than including static template text. For performance reasons, it is best to reserve the use of the `out.print()` command for dynamic text.

Example 6–1 and Example 6–2 are two HTML coding examples. For these JSP samples, Example 6–2 should be more efficient and give better performance.

Example 6–1 Using out.print

```
<%
  out.print("<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>\n");
  out.print("<BODY BGCOLOR=\"#ffffff\">\n");
  out.print("<H1 ALIGN=\"center\">Book Store Web Commerce Test</H1>\n");
  out.print("<P ALIGN=\"CENTER\">\n");
  out.print("<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
              "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>\n");
  out.print("<H2 ALIGN=\"center\">Home Page</H2>\n");
%>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

Example 6–2 Using Static Text

```
<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>
<BODY BGCOLOR=\"#ffffff\">
<H1 ALIGN=\"center\">Bookstore Web Commerce Test </H1>
<P ALIGN=\"CENTER\">
<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
         "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>
<H2 ALIGN=\"center\">Home Page</H2>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

Performance Issues for Buffering JSPs

By default, a JSP uses an area of memory known as a page buffer. The page buffer, set to 8KB by default, is required if the JSP uses dynamic globalization, `contextType` settings, error pages, or forwards. If the page does not use these features, then you can disable buffering with the following command:

```
<%@ page buffer="none" %>
```

Disabling buffering by setting the buffer value to `none` improves the performance of the page by reducing memory usage and saving the processing step of copying the buffer.

When you need buffering, it is important to select an adequate size for your buffer. If you are writing a page that is larger than the default 8KB buffer, and you have not reset the buffer size, then the JSP `autoflush` will be activated which could have performance implications. Therefore, if buffering is necessary for your JSP, make sure to set the page buffer to an appropriate size. For example, to set the buffer size to 24KB, use the following command:

```
<%@ page buffer="24KB" %>
```

Using Static Versus Dynamic Includes

The `include` directive makes a copy of the included page and copies it into a JSP (`including page`) during translation. This is known as a **static include** (or **translate-time include**) and uses the following syntax:

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

Alternatively, the `jsp:include` tag dynamically includes output from the included page within the output of the including page, during runtime. This is known as a **dynamic include** (or **runtime include**) and uses the following syntax:

```
<jsp:include page="/jsp/userinfopage.jsp" flush="true" />
```

If you have static text, that is not too large, for performance reasons, it is better to use a static include rather than a dynamic include.

In general, when working with includes, note the following:

- Static includes affect page size. Static includes avoid the overhead of the request dispatcher that a dynamic include necessitates, but may be problematic where large files are involved. Static includes are typically used to include small files whose content is used repeatedly in multiple JSPs. For example:
  - Statically include a logo or copyright message at the top or bottom of each page in your application.
  - Statically include a page with declarations or directives, such as imports of Java classes, that are required in multiple pages.
  - Statically include a central `status checker` page from each page of your application.
- Dynamic includes affect processing overhead and performance. Dynamic includes are useful for modular programming. You may have a page that sometimes executes on its own but sometimes is used to generate some of the

output of other pages. Dynamically included pages can be reused in multiple including pages without increasing the size of the including pages.

Note: Both static includes and dynamic includes can be used only between pages in the same servlet context.

See Also: *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* in the Oracle9*i* Application Server documentation library

Performance Issues for Including Static Content

JSPs containing a large amount of static content, including large amounts of HTML code that does not change at runtime, may result in slow translation and execution.

There are two workarounds for this issue that may improve performance:

- Put the static HTML into a separate file and use a dynamic `include` command (`jsp:include`) to include its output in the JSP output at runtime.

  Note: A static `<%@ include... %>` command would not work. It would result in the included file being included at translation time, with its code being effectively copied back into the including page. This would not solve the problem.

- Put the static HTML into a Java resource file.

  The JSP translator will do this for you if you enable the `external_resource` configuration parameter.

  For pre-translation, the `-extres` option of the `ojspc` tool also offers this functionality.

  Note: Putting static HTML into a resource file may result in a larger memory footprint than the `jsp:include` workaround mentioned above, because the page implementation class must load the resource file whenever the class is loaded.

Improving EJB Performance in Oracle9*i*AS

This section covers configuration parameters that you set to control how OC4J handles EJBs. Tuning these options can improve the performance of EJBs running on OC4J.

This section includes the following topics:

- Setting server.xml Configuration Parameters for EJBs
- Setting OC4J Specific Configuration Parameters for EJBs

Setting server.xml Configuration Parameters for EJBs

This section covers parameters that you can tune for EJB performance in the `server.xml` file for an OC4J Instance.

Setting the Transaction Configuration Timeout

You can change the default value for the transaction configuration timeout in the `transaction-config` element in the `server.xml` file for the OC4J Instance. The `transaction-config` timeout is not an EJB specific timeout, but affects all transactions which use EJBs. This configuration parameter specifies the maximum time taken for a transaction to finish before it can get rolled back due to a timeout. This parameter applies to all transactions on the OC4J Instance. The default value for this parameter is 60000 milliseconds (60 seconds).

For example, the following `server.xml` sets the `transaction-config` timeout parameter to 30 seconds:

```
<transaction-config timeout="30000"/>
```

Increase this value if your applications that use transactions are getting transaction timeout errors, or if your transactions may be longer than 60 seconds (including waiting for connections set by `wait-timeout` in `datasources.xml`).

The `transaction-config` timeout applies for all transactions running in OC4J, and therefore must be big enough for your longest transaction. If you specify a small timeout value for `transaction-config` timeout, then you cannot set the timeout to a larger value for an individual EJB, since the `transaction-config timeout` applies for all transactions at the EJB level. Thus, the timeout should be set to a value greater than or equal to the timeouts used within a transaction (for example the data sources `wait-timeout`, and the EJB `call-timeout`).

See Also:

- "Setting the Wait for Free Connection Timeout in Data Sources" on page 6-14
- "Configuring Parameters that Apply for All EJBs" on page 6-29

Setting OC4J Specific Configuration Parameters for EJBs

This section covers parameters that you can tune for EJB performance that are specific to OC4J. These parameters are set in the `orion-ejb-jar.xml` file.

This section covers the following topics:

- Configuring Parameters that Apply for All EJBs
- Configuring Parameters for CMP Entity Beans
- Configuring Parameters for BMP Entity Beans
- Configuring Parameters for Session Beans

Configuring Parameters that Apply for All EJBs

Table 6–4 lists parameters that you can tune for EJB performance that are specific to OC4J. These parameters apply for all types of EJBs, including session and entity beans. The parameters in Table 6–4 are specified in `orion-ejb-jar.xml`.

Table 6–4 EJB Parameters That Apply for All EJB Types

Parameter	Description
`call-timeout`	Applies for session and entity beans. This parameter specifies the maximum time to wait for any resource that the EJB container needs, excluding database connections, before the container calls the EJB method. Default Values: 90000 milliseconds for entity beans and 0 (forever) for session beans. See Also: "Setting the Transaction Configuration Timeout" on page 6-28

Table 6–4 (Cont.) EJB Parameters That Apply for All EJB Types

Parameter	Description
`max-tx-retries`	Applies for session and entity beans. This parameter specifies the number of times to retry a transaction that was rolled back due to system-level failures. Generally, we recommend that you start by setting `max-tx-retries` to 0 and adding retries only where errors are seen that could be resolved through retries. For example, if you are using serializable isolation and you want to retry the transaction automatically if there is a conflict, you might want to use retries. However, if the bean wants to be notified when there is a conflict, then in this case, you should set `max-tx-retries=0`. Default Value: 3 (for session beans and entity beans) See Also: "Setting the Transaction Configuration Timeout" on page 6-28 See Also: "Setting the Connection Retry Interval in Data Sources" on page 6-14

Configuring Parameters for CMP Entity Beans

This section covers parameters for entity beans using CMP. These parameters are specified in the `orion-ejb-jar.xml` configuration file that affect performance.

Table 6–5 lists the entity bean CMP specific parameters.

Table 6–6 describes the supported `locking-mode` parameter values.

Table 6–5 CMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`do-select-before-insert`	Recommend setting to `false` to avoid the extra select before insert which checks if the entity already exists before doing the insert. This will then detect a duplicate, if there is one, during the insert. Default Value: true
`exclusive-write-access`	Default is false for beans with `locking-mode=optimistic` or `pessimistic` and `true` for `locking-mode=read-only`. This parameter corresponds to which commit option is used (A, B or C, as defined in the EJB specification). When `exclusive-write-access = true`, this is commit option A. The `exclusive-write-access` is forced to `false` if locking is pessimistic or optimistic, and is not used with EJB clustering. The `exclusive-write-access` can be false with read-only locking, but read-only won't have any performance impact if `exclusive-write-access=false`, since ejbStores are already skipped when no fields have been changed. To see a performance advantage and avoid doing ejbLoads for read-only beans, you must also set exclusive-write-access=true.

Table 6–5 (Cont.) CMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`isolation`	If your database is already configured with the isolation mode you want for your transactions, you'll get better performance if you don't explicitly set the isolation mode attribute in the `orion-ejb-jar.xml file`. Omitting the isolation setting means to use the database default setting, and extra processing will not be done to explicitly set isolation levels in your transactions. See Table 6–7 for a description of `isolation` options and how they relate to locking modes. Default value:
`locking-mode`	The locking modes, specified with the `locking-mode` parameter, manage concurrency and configure when to block to manage resource contention or when to execute in parallel. See Table 6–6 for a description of `locking-mode`. See Table 6–7 for a description of `isolation` options and how they relate to locking modes.
`max-tx-retries`	For a description, see Table 6–4
`update-changed-fields-only`	Specifies whether the container updates only modified fields or all fields to persistence storage for CMP entity beans when ejbStore is invoked. Default Value: `true`
`validity-timeout`	The `validity-timeout` is only used when `exclusive-write-access=true` and `locking-mode=read-only`. The validity timeout is the maximum time in milliseconds that an entity is valid in the cache (before being reloaded). We recommend that if the data is never being modified externally (and therefore you've set `exclusive-write-access=true`), that you can set this to 0 or -1, to disable this option, since the data in the cache will always be valid for read-only EJBs that are never modified externally. If the EJB is generally not modified externally, so you're using `exclusive-write-access=true`, yet occasionally the table is updated so you need to update the cache occasionally, then set this to a value corresponding to the interval you think the data may be changing externally.

Note: The following CMP entity bean parameters are not supported in this release:

`max-instances`, `min-instances`, `pool-cache-timeout`

Table 6–6 CMP Entity Bean Locking-Mode Values

Locking Mode Value	Description
`optimistic`	Multiple users can execute the entity bean in parallel. The optimistic locking mode does not monitor resource contention; thus, the burden of the data consistency is placed on the database isolation modes. This is the default value for `locking-mode`.
`pessimistic`	Manages resource contention and does not allow parallel execution. Only one user at a time is allowed to execute the entity bean. Pessimistic locking uses "SELECT....FOR UPDATE" to serialize access in the database.
`read-only`	Multiple users can execute the entity bean in parallel. The container does not allow any updates to the bean's state.

The `locking-mode`, along with `isolation`, assures database consistency for EJB entity beans using CMP. Table 6–7 shows the common `locking-mode` and `isolation` combinations. The different combinations have both functional and performance implications, but often the functional requirements for data consistency will lead to selecting a mode, even when it may be at the expense of performance.

Table 6–7 CMP Entity Bean Locking-Mode and Isolation Relationships

Locking-mode	Isolation	When to Use
`pessimistic`	`committed`	If data consistency must be guaranteed, and frequent concurrent updates to the same rows are expected.
`pessimistic`	`serializable`	We recommend that this combination not be used.
`optimistic`	`committed`	If concurrent reads and updates to the same rows with read-committed semantics is sufficient.
`optimistic`	`serializable`	If data consistency must be guaranteed, but infrequent concurrent updates to the same rows are expected.
`read-only`	`committed`	If repeatable read is not required.
`read-only`	`serializable`	If repeatable read is required.

In Table 6–7 the `isolation` setting refers to either the transaction `isolation` attribute setting, if explicitly set, or to the database isolation level (if the transaction `isolation` attribute is not set). Also, although `locking-mode` and transaction

isolation levels are set as attributes of a CMP bean, the isolation level that will be in effect for the transaction is the isolation level of the first entity bean used in the transaction. Therefore it is best to set all beans in the same transaction to the same isolation level.

In general, optimistic locking with committed isolation will be faster since it allows for more concurrency, but it may not meet your needs for data consistency. Pessimistic locking with committed isolation, and optimistic locking with serializable isolation will be slower, but will guarantee data consistency on updates.

Defining a bean as read-only will assure that no updates are allowed to the bean. The performance will be similar to a bean which may not be defined as read-only, and yet is never used to do inserts, updates, or deletes (i.e. only the methods which read are called). This is because if no fields are modified in a bean that is not defined with read-only locking, it is already optimized to not do an ejbStore. To see a performance advantage and avoid doing ejbLoads for read-only beans, you must also set `exclusive-write-access=true`.

Configuring Parameters for BMP Entity Beans

This section covers parameters that apply to entity beans using BMP. These parameters are specified in the `orion-ejb-jar.xml` configuration file.

Table 6–8 lists the entity bean BMP specific parameters.

Table 6–8 BMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`locking-mode`	The locking modes, specified with the `locking-mode` parameter, manage concurrency and configure when to block to manage resource contention or when to execute in parallel. BMP beans must use optimistic locking, which allows concurrent access to a bean, and the BMP bean is responsible for managing the database access and data consistency. It is up to the BMP bean to manage isolation as well, and therefore the isolation settings do not apply for BMP Default Value: `optimistic`
`max-tx-retries`	For a description, see Table 6–4

Note: The following BMP entity bean parameters are not supported in this release.

`max-instances`, `min-instances`, `pool-cache-timeout`

Configuring Parameters for Session Beans

This section covers the parameters that are specified in the `orion-ejb-jar.xml` configuration file and apply for session beans.

Table 6–9 lists the stateless session bean specific parameters.

Table 6–10 lists the stateful session bean specific parameters.

Table 6–9 Stateless Session Bean Parameters

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`cache-timeout`	The `cache-timeout` applies for Stateless Session EJBs. This parameter specifies how long to keep stateless sessions cached in the pool. For stateless session beans, if you specify a `cache-timeout`, then at every `cache-timeout` interval, all beans in the pool, of the corresponding bean type, are removed. If the value specified is zero or negative, then the `cache-timeout` is disabled and beans are not removed from the pool. Default Value: 60 (seconds)
`max-tx-retries`	For a description, see Table 6–4

Table 6–10 Stateful Session Bean Parameters

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`timeout`	The `timeout` applies for Stateful Session EJBs. If the value is zero or negative, then all timeouts are disabled. The timeout parameter is an inactivity timeout for stateful session beans. Every 30 seconds the pool clean up logic is invoked. Within the pool clean up logic, only the sessions that timed out, by passing the `timeout` value, are deleted. Adjust the `timeout` based on your applications use of stateful session beans. For example, if stateful session beans are not removed explicitly by your application, and the application creates many stateful session beans, then you may want to lower the `timeout` value. If your application requires that a stateful session bean be available for longer than 30 minutes, then adjust the timeout value accordingly. Default Value: 30 (minutes)
`max-tx-retries`	For a description, see Table 6–4

Using Multiple OC4Js and Limiting Connections

This section describes techniques that allow you to improve performance by setting the number of active processes within an OC4J Instance, sending applications to different OC4J Instances, and limiting the number of requests sent to an OC4J Instance.

This section covers the following topics:

- Limiting HTTP Connections
- Configuring Multiple OC4J Processes
- Balancing Applications Across OC4J Instances

Limiting HTTP Connections

You can improve J2EE application performance by limiting the number of active HTTP concurrent connections a given site accepts. Using Oracle HTTP Server with `mod_oc4j`, you can limit the number of incoming requests by setting the `MaxClients` parameter in `httpd.conf`.

> **See Also:** "Configuring Oracle HTTP Server Directives" on page 5-10

Limiting HTTP Connections with Standalone OC4J

If you are using standalone OC4J you can limit the number of active web users an OC4J site accepts concurrently by constraining the maximum allowable HTTP connections. Tuning parameters on a standalone OC4J can improve performance if there are a large number of concurrent users that the system cannot efficiently handle, or when there are limited resources which you cannot easily constrain.

To limit the HTTP connections, use the `max-http-connections` configuration element in `server.xml` and specify the attributes: `value`, `max-connections-queue-timeout`, and `socket-backlog` attributes.

For example, the following shows a line of server.xml that configures the maximum number of connections:

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100"/>
```

Table 6–11 describes the `max-http-connections` attributes.

When you want messages to be redirected to a different URL when the maximum connections limit is reached, include the HTTP redirect URL.

For example, to redirect to `http://optional.redirect.url/page.jsp`, add the following line to `server.xml`:

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100">http://optional.redirect.url/page.jsp</max-http-connections>
```

Table 6–11 Setting max-http-connections Attributes

Attribute	Description
`max-connections-queue-timeout`	Specifies the number of seconds to wait for an available connection if the maximum connections have been reached before returning either server busy or redirect message. Default Value: 10 (seconds)
`socket-backlog`	Specifies the number of connections that can be queued up before denying connections at the socket level. Default Value: 30
`value`	Specifies the maximum number of connections. Default Value: 100000

See Also: Appendix B, "Additional Information" in the *Oracle9iAS Containers for J2EE User's Guide* for information on the elements in OC4J `server.xml` file.

Configuring Multiple OC4J Processes

Oracle9*i*AS lets you configure multiple OC4J processes in an OC4J Instance. If there are sufficient resources on your system, you may be able to improve performance by configuring multiple OC4J processes for the OC4J Instance that services your J2EE application. Using multiple OC4J processes is the simplest form of load balancing without using Oracle9*i*AS clustering. Configuring an OC4J Instance to allow requests to be served by multiple OC4J processes can reduce or avoid contention and improve scalability. Oracle9*i*AS also supports multi node clustering for load balancing.

The optimal ratio of OC4J processes to CPUs is dependent on characteristics of the application, hardware configuration and JDK being used. But in general, for multi-cpu configurations with greater than two processors, you should consider configuring multiple OC4J processes. For example, on a recent test of a J2EE application, a single OC4J process was sufficient to use 70% of the resources on a four processor system. That statistic indicated that either adding a second OC4J process would be advantageous or that the incoming load was insufficient to utilize more of the machine resources. By monitoring the incoming load, we were able to determine that it was the former, and adding a second process improved performance.

Adding processes beyond the available resources of the system will not improve performance. For example, if one OC4J process is sufficient to saturate the CPU resources of a system, adding four more processes is not likely to improve performance and may, in fact, degrade it. A good starting point is to configure one OC4J process for every 2-3 CPUs and measure the improvement derived from adding additional processes.

Configuring Multiple OC4J Processes Using Oracle Enterprise Manager

Using Oracle Enterprise Manager you can specify the number of processes in an OC4J Instance using the OC4J Instance level configuration available on the OC4J Instance's home page.

If you configure your OC4J to use load-balancing with multiple OC4J processes running within the same island, and you do not want sessions to be replicated, make sure that the `<distributable/>` property is not set in the application's `web.xml` file.

Note: Setting the `<distributable/>` property in `web.xml` can have significant performance overhead for applications that use sessions, since this configures the application to use session replication with failover when multiple OC4J processes are running within the same island.

See Also: Chapter 3, "Advanced Configuration, Development, and Deployment", in *Oracle9iAS Containers for J2EE User's Guide*

Balancing Applications Across OC4J Instances

It is often beneficial to spread J2EE application load among multiple OC4J Instances, especially when applications are run on a multiprocessor system. Running multiple OC4J Instances generally results in higher **throughput** and shorter **response time**, even on a single-processor host.

If your web site has many different applications deployed which have different requirements and needs, you may want to configure different OC4J Instances, each with the appropriate number of OC4J processes to service the different applications.

To deploy applications to different OC4J Instances, perform the following steps:

1. Create the multiple OC4J Instances.
2. Using the Application Deployment Wizard, for each Instance, deploy the appropriate application to each instance, and specify a unique URL mapping for each of the applications.

After deploying the applications to different OC4J Instances, you can monitor the performance to see if throughput increases, or the response time decreases.

Database Monitoring and Tuning

To achieve optimal performance in Oracle9*i*AS OC4J J2EE applications that use the database, the database tables you access need to be designed with performance in mind, and you need to monitor and tune the database server to assure that the system is performant.

See Also: *Oracle9i Database Performance Tuning Guide and Reference*

Improving BC4J Performance in Oracle9*i*AS

This section contains tips for improving the maintainability, scalability, and performance of your Oracle Business Components for Java (BC4J) applications.

> **See Also:** *Developing Business Components* under the heading Oracle9*i*AS Business Components for Java in the Oracle9*i*AS documentation library

Choose the Right Deployment Configuration

Your application will have the best performance and scalability if you deploy your business components to the web module with your client. Unless you have strong reasons (such as wanting to use distributed transactions or EJB security features), we recommend web module deployment of business components over EJB deployment.

Note that both web module deployment and EJB deployment are fully J2EE-compliant, and the BC4J framework makes it easy to switch between them. You can test your application in both modes to see which gives you the best performance.

Use Application Module Pooling for Scalability

A client can use application module instances from a pool, called application module pooling. This offers these advantages:

- It reduces the amount of time to obtain server-side resources
- It allows a small number of instances to serve a much larger number of requests
- It addresses the requirements of web applications that must handle thousands of incoming requests
- It lets you preserve session state and provides failover support

For example, in the case of a web application, you may have 1,000 users but you know that only 100 will be using a certain application module at one time. So you use an application module pool. When a client needs an application module instance, it takes a free one from the pool and releases it to the pool after either committing or rolling back the transaction. Because the instance is precreated, end users are saved the time it takes to instantiate the application module when they want to perform a task. Typically, web-based JSP clients use pools. If you want to make sure that the application module pool has a maximum of 100 application module instances, you can customize the default application module pool.

If your client needs to keep track of application module state, we recommend using stateful mode. In a stateful JSP application, the client does not reserve the application module instance, making it available to other clients if the number of application modules exceeds the recycle threshold. State is, instead, preserved in one of two ways: The application module pool returns a client's original application module if the application module has not been recycled, and the pool persists the state of recycled application modules in the database to be available to clients that request them later.

When you release an application module at the end of a user's session, be sure to use stateless (rather than stateful or reserved) release mode. This frees up database space and allows the pool to recycle the application module immediately.

Perform Global Framework Component Customization Using Custom Subclasses

Particularly in large organizations, you may want specific functionality shared by all components of a particular type--for example, by all view objects. An architect can create a thin layer of classes such as `MyOrgViewObjectImpl` that implement the desired behavior. Individual developers can extend `MyOrgViewObjectImpl` instead of `ViewObjectImpl`, and you can use the "substitutes" feature to extend `MyOrgViewObjectImpl` in legacy code.

Use SQL-Only and Forward-Only View Objects when Possible

Basing a view object on an entity object allows you to use the view object to insert, update, and delete data, and helps keep view objects based on the same data synchronized. However, if your view object is only going to be used for read-only queries, and there is no chance that the data being queried in this view object will have pending changes made through another view object in the same application module, you should use a SQL-only view object that has no underlying entities. This will give you improved performance, since rows do not need to be added to an entity cache.

If you are scrolling through data in one direction, such as formatting data for a web page, or for batch operations that proceed linearly, you can use a forward-only view object. Forward-only mode prevents data from entering the view cache. Using forward only mode can save memory resources and time, because only one view row is in memory at a time. Note that if the view object is based on one or more entity objects, the data does pass to the entity cache in the normal manner, but no rows are added to the view cache.

Do Not Let Your Application Modules Get Too Large

A root application module should correspond to one task--anything that you would include in a single database transaction. Do not put more view objects or view links than you will need for a particular task in a single application module.

In addition, consider deferring the creation of view links by creating them dynamically with `createViewLink()`. If you include all view links at design time, the business logic tier will automatically execute queries for all detail view objects when your client navigates through a master view object. Deferring view link creation will prevent the business logic tier from executing queries for detail view objects that you do not yet need.

For example, for a form in which detail rows are displayed only on request (rather than automatically), including a view link at design time would force the business logic tier to automatically execute a query that might well be unnecessary. To prevent this, you should create a view link dynamically when the detail rows are requested. By contrast, for a form in which detail rows are displayed as soon as a master is selected, you should use a view link created at design time to avoid the runtime overhead of calling `createViewLink()`.

Use the Right Failover Mode

By default, the application module pool supports failover, which saves an application module's state to the database as soon as the application module is checked into the pool. If the business logic tier or the database becomes inoperable in mid-transaction (due to a power failure or system crash, for example), the client will be able to instantiate a new application module with the same state as the lost one, and no work will be lost.

However, some applications do not require this high level of reliability. If you're not worried about loss of work due to server problems, you may want to disable failover. When failover is disabled, the application module's state exists only in memory until it is committed to the database (at which point the application module's state is discarded) or recycled (at which point the state is saved so that the client can retrieve it). By not saving the application module state every time the application module is checked in, failover-disabled mode can improve performance.

Use View Row Spillover to Lower the Memory to Cache a Large Number of Rows

While the business logic tier is running, it stores view rows in a cache in memory (the Java heap). When the business logic tier needs to store many rows at once, you

need to make sure it doesn't run out of memory. To do so, you can specify that when the number of rows reaches a certain size, the rows "overflow" to your database to be stored on disk. This feature is called view row spillover. If your application needs to work with a large query result, view row spillover can help the cache operate more efficiently.

Choose the Right Style of Bind Parameters

Oracle-style bind parameters (:1, :2, and so on) are more performant than JDBC-style bind parameters.

There are only two reasons to use JDBC-style bind parameters:

- Use JDBC-style bind parameters if you may use a non-Oracle JDBC driver.
- Use JDBC-style bind parameters if you have more than one occurrence of the same parameter in the `WHERE` clause.

Implement Query Conditions at Design Time if Possible

You should include any portion of your query condition that you know in advance in the `WHERE` clause field in the View Object wizard. Only use `setWhereClause()` for genuinely dynamic query conditions.

Even if your query conditions are genuinely dynamic, you may be able to use parametrized queries instead of `setWhereClause()`. For example, if your view object needs to execute a query with the `WHERE` clause `EMPLOYEE_ID=<x>` for various values of x, use a parametrized `WHERE` clause such as `EMPLOYEE_ID=:1`. This is more efficient than repeatedly calling `setWhereClause()`.

Use the Right JDBC Fetch Size

The default JDBC fetch size is optimized to provide the best tradeoff between memory usage and network usage for many applications. However, if network performance is a more serious concern than memory, consider raising the JDBC fetch size.

Turn off Event Listening in View Objects used in Batch Processes

In non-interactive, batch processes, there is no reason for view objects to listen for entity object events. Use `ViewObject.setListenToEntityEvents(false)` on such view objects to eliminate the performance overhead of event listening.

Exhibit 15

Oracle® Application Server Containers for J2EE

Support for JavaServer Pages Developer's Guide

10*g* Release 2 (10.1.2)

B14014-02

July 2005

ORACLE

Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide, 10g Release 2 (10.1.2)

B14014-02

Copyright © 2000, 2005, Oracle. All rights reserved.

Primary Author: Dan Hynes, Brian Wright

Contributing Author: Michael Freedman

Contributors: Ashok Banerjee, Ellen Barnes, Julie Basu, Matthieu Devin, Jose Alberto Fernandez, Sumathi Gopalakrishnan, Ralph Gordon, Ping Guo, Hal Hildebrand, Susan Kraft, Sunil Kunisetty, Clement Lai, Song Lin, Jeremy Lizt, Angie Long, Sharon Malek, Sheryl Maring, Kuassi Mensah, Jasen Minton, Kannan Muthukkaruppan, John O'Duinn, Robert Pang, Olga Peschansky, Shiva Prasad, Jerry Schwarz, Sanjay Singh, Gael Stevens, Kenneth Tang, YaQing Wang, Alex Yiu, Shinji Yoshida, Helen Zhao

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

2

Overview of the Oracle JSP Implementation

The JSP container provided with Oracle Application Server Containers for J2EE (OC4J) in the Oracle Application Server is a complete implementation of the JSP 1.2 specification. This functionality depends upon servlet 2.3 functionality, and the OC4J servlet container is a complete implementation of the servlet 2.3 specification.

This chapter provides overviews of the Oracle Application Server, OC4J, the OC4J JSP implementation and features, and custom tag libraries and utilities that are also supplied (documented in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*).

The following sections are included:

- Overview of the Oracle Application Server and JSP Support
- Oracle JDeveloper JSP Support
- Overview of Oracle Value-Added Features

Overview of the Oracle Application Server and JSP Support

The following sections provide a brief overview of the Oracle Application Server, its J2EE environment, its JSP implementation, and its Web server:

- Overview of the Oracle Application Server
- Oracle HTTP Server and mod_oc4j
- Overview of OC4J
- Overview of the JSP Implementation in OC4J

> **Note:** Users of earlier Oracle Application Server releases can refer to *Oracle Application Server Upgrade and Compatibility Guide* for information about issues in migrating to the current release.

Overview of the Oracle Application Server

Oracle Application Server is a scalable, secure, middle-tier application server. It can be used to deliver Web content, host Web applications, connect to back-office applications, and make these services accessible to any client browser. Users can access information, perform business analysis, and run business applications on the Internet or corporate intranets or extranets. Major areas of functionality include business intelligence, e-business integration, J2EE Web services, performance and caching, portals, wireless services, and management and security. For performance, scalability, and dependability, there are also clustering and load-balancing features.

To deliver this range of content and services, the Oracle Application Server incorporates many components, including the Oracle HTTP Server, Oracle Application Server Web Cache, Oracle Application Server Web Services, Oracle Application Server Portal, Oracle Application Server Wireless, and business logic runtime environments that support Enterprise JavaBeans, stored procedures, and Oracle Application Development Framework (Oracle ADF) Business Components.

For its Java environment, Oracle Application Server provides the Oracle Application Server Containers for J2EE (OC4J), a J2EE 1.3-compliant set of containers and services. This includes the JSP container described in this manual, a servlet container, and an EJB container.

For administration, you can fully manage and configure Oracle Application Server and OC4J using the HTML-based Oracle Enterprise Manager 10*g*. This includes full support for managing clustering, configuration, and deployment.

Oracle HTTP Server and mod_oc4j

Oracle HTTP Server, powered by the Apache Web server, is included with Oracle Application Server as the HTTP entry point for Web applications, particularly in a production environment. By default, it is the front-end for all OC4J processes. Client requests go through Oracle HTTP Server first.

When the Oracle HTTP Server is used, dynamic content is delivered through various Apache *mod* components provided either by the Apache Software Foundation or by Oracle. Static content is typically delivered from the file system, which is more efficient in this case. An Apache mod is typically a module of C code, running in the Apache address space, that passes requests to a particular mod-specific processor. The mod software will have been written specifically for use with the particular processor.

Oracle Application Server supplies the `mod_oc4j` Apache mod, which is used for communication between the Oracle HTTP Server and OC4J. It routes requests from the Oracle HTTP Server to OC4J processes, and forwards responses from OC4J processes to Web clients.

Communication is through the Apache JServ protocol (AJP). AJP was chosen over HTTP because of a variety of AJP features allowing faster communication, including use of binary format and more efficient processing of message headers.

The following features are provided with `mod_oc4j`:

- Load balancing capabilities across many back-end OC4J clusters
- Stateless session routing of stateful servlets

  This is accomplished through enhanced use of cookies. Routing information is maintained in the cookie itself to ensure that stateful servlets are always routed to the same OC4J JVM.

> **Note:** Oracle HTTP Server is not relevant for an OC4J standalone environment (typically used only during development). See "OC4J Standalone" on page 2-5.

Overview of OC4J

The following sections provide an overview of features of OC4J, the J2EE component of the Oracle Application Server:

- OC4J General Features

ORCL00444378

- OC4J Services
- OC4J Containers
- OC4J Standalone

OC4J General Features

OC4J is a high-performance, J2EE-compliant set of containers and services providing a scalable and reliable server infrastructure. In Oracle Application Server 10*g* Release 2 (10.1.2), OC4J complies with the J2EE 1.3 specification.

For developer convenience, OC4J has been integrated with Oracle JDeveloper and other development tools, and can run in a standalone mode separate from Oracle Application Server during the development process.

Java applications built with any development tool can be deployed to OC4J, which supports standard EAR or WAR file deployment. You can debug applications deployed to OC4J through standard Java profiling and debugging facilities.

For security, OC4J supports Secure Socket Layer (SSL) and HTTPS functionality.

> **Note:** Each OC4J instance runs in a single Java virtual machine.

OC4J Services

OC4J supports the following Java and J2EE services:

- J2EE Connector Architecture (JCA): JCA defines a standard architecture for connecting J2EE platforms to heterogeneous enterprise information systems such as ERP systems, mainframe transaction processing, database systems, and legacy applications.
- Java Transaction API (JTA) and two-phase commits: JTA allows simultaneous updates to multiple resources in a single, coordinated transaction.
- Java Message Service (JMS) integration: This integration allows compatibility between the Oracle JMS implementation and those of other JMS providers.
- Java Naming and Directory Interface (JNDI): JNDI associates names with resources for lookup purposes.
- Java Authentication and Authorization Service (JAAS): The Oracle implementation of JAAS and the Java2 security model provides complete support for development and deployment of secure applications and for fine-grained authorization and access control.
- JDBC data sources: This is the standard mechanism for connecting to a database.

See the *Oracle Application Server Containers for J2EE Services Guide* for information.

OC4J Containers

The OC4J 10.1.2 implementation supplies the following J2EE containers:

- A JSP container complying with the Sun JSP 1.2 specification

  The JSP bundle also supplies tag libraries to implement Web services, caching capabilities, SQL access, file access, and other features. For further overview of the JSP container provided with OC4J, see "Overview of the JSP Implementation in OC4J" on page 2-5.

ORCL00444379

- A servlet container complying with the servlet 2.3 specification (with key features described immediately below)
- An EJB container complying with the EJB 2.0 specification (with key features described below)

OC4J containers have been instrumented to support the Dynamic Monitoring Service (DMS) to provide runtime performance data. You can view this data through Enterprise Manager.

> **Note:** Servlet 2.3 compliance is required in order to support JSP 1.2 compliance.

Key Servlet Container Features The OC4J servlet container supports the following key features:

- SSL and HTTPS: In Oracle Application Server, OC4J supports SSL (Secure Socket Layer) communication between Oracle HTTP Server and OC4J, using secure AJP. In addition, OC4J standalone supports SSL communication directly between a client and OC4J, using HTTPS.
- Integration with SSO and Oracle Internet Directory: This is through the Oracle JAAS implementation.
- Stateful failover and cluster deployment: For a distributable application, session state is replicated to alternate OC4J servers so that state is not lost in the event of failover.
- Servlet filtering: This allows transformation of the content of an HTTP request or response, and modification of header information.
- Application-level and session-level event listeners: This feature allows greater control over interaction with servlet context and HTTP session objects and, therefore, greater efficiency in managing resources that the application uses.

See the *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for information.

Key EJB Container Features The OC4J EJB container supports the following:

- Session beans: A session bean is used for task-oriented requests. You can define a session bean as stateless or stateful. Stateless session beans cannot maintain state information across calls, while stateful session beans *can* maintain state across calls.
- Entity beans: An entity bean represents data. It can use the container to maintain the data persistently, which is referred to as *container-managed persistence* (CMP), or it can use the bean implementation to manage the data, which is referred to as *bean-managed persistence* (BMP).
- Message-driven beans (MDB): A message-driven bean is used to receive JMS messages from a queue or topic. It can then invoke other EJBs to process the JMS message.

EJB support in OC4J also includes these features:

- Clustering for session and entity beans
- Enhanced entity bean concurrency models to support concurrent access from multiple clients

ORCL00444380

- Extended locking models for entity beans (optimistic locking mode / pessimistic locking mode / read-only mode)
- Active Components for Java (AC4J), to provide a standards-based infrastructure for coordinating long-running business transactions

See the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for information.

OC4J Standalone

In a production environment, it is typical to use OC4J inside a complete Oracle Application Server environment, including the Oracle HTTP Server (as described in "Oracle HTTP Server and mod_oc4j" on page 2-2), OracleAS Web Cache, and Enterprise Manager.

For a development environment, OC4J is also available as a standalone component by downloading `OC4J_extended.zip` from the Oracle Technology Network at the following location:

```
http://www.oracle.com/technology/tech/java/oc4j
```

When using OC4J standalone, you can use its own HTTP Web listener through port 8888. For information about OC4J standalone, see the *Oracle Application Server Containers for J2EE Stand Alone User's Guide* (downloadable with OC4J_extended.zip) and the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*.

> **Note:** To use OC4J standalone, you must have a supported version of the Sun Microsystems JDK installed. A JDK is not provided with the OC4J standalone product.

Overview of the JSP Implementation in OC4J

The JSP container in Oracle Application Server 10*g* Release 2 (10.1.2) is compliant with the JSP 1.2 specification.

In general, a JSP 1.2 environment requires a servlet 2.3 environment, such as the OC4J servlet container. The OC4J JSP implementation, however, also supports running on a servlet 2.0 environment. To make this possible, the OC4J JSP container emulates required servlet features beyond the 2.0 specification.

For a variety of reasons, though, it is generally advisable to use the OC4J servlet 2.3 environment.

These features are discussed in the following sections:

- History and Integration of JSP Containers
- JSP Front-End Servlet and Configuration
- OC4J JSP Features for JSP 1.2
- Configurable JSP Extensions in OC4J
- Portability Across Servlet Environments

History and Integration of JSP Containers

In Oracle9*i*AS Release 1.0.2.2, the first release to include OC4J, there were two JSP containers: 1) a container developed by Oracle and known as "OracleJSP"; 2) a container licensed from Ironflare AB and known as the "Orion JSP container".

ORCL00444381

The OracleJSP container offered several advantages, including useful value-added features and enhancements such as for globalization. The Orion container also offered advantages, including superior speed, but had disadvantages as well. It did not always exhibit standard behavior when compared to the JSP reference implementation (Tomcat), and its support for internationalization and globalization was not as complete.

Oracle9*i*AS Release 2 (9.0.2) first integrated the OracleJSP and Orion containers into a single JSP container referred to in this manual as the "OC4J JSP container". This container offers the best features of both previous versions, runs efficiently as a servlet in the OC4J servlet container, and is integrated with other OC4J containers as well. The integrated container primarily consists of the OracleJSP translator and the Orion container runtime, running with a simplified dispatcher and the OC4J core runtime classes.

JSP Front-End Servlet and Configuration

The JSP container in OC4J uses the front-end servlet `oracle.jsp.runtimev2.JspServlet`. See "JSP Configuration in OC4J" on page 3-10.

For non-OC4J environments, use the old front-end servlet, `oracle.jsp.JspServlet`.

OC4J JSP Features for JSP 1.2

In the OC4J 10.1.2 implementation, the OC4J JSP container is fully compliant with the JSP 1.2 specification. Most functionality introduced in this specification is in the area of custom tag libraries.

- Tag library features:
  - There is a tag handler interface that allows iteration through a tag body without having to maintain and access a body content object.
  - You can create a tag-library-validator class and associate it with a tag library. A validator instance will check any JSP page that uses the library, to verify that it meets whatever constraints you desire.
  - For convenience, you can declare servlet event listeners in a tag library descriptor file instead of in the `web.xml` file. This enables you to more conveniently manage application and session resources associated with usage of the tag library.
  - You can package multiple tag libraries and their TLD files inside a single JAR file.

  See Chapter 8, "JSP Tag Libraries" for details about these features, and "Overview of Tag Library Changes Between the JSP 1.1 and 1.2 Specifications" on page 8-3 for a more detailed summary.
- XML features:
  - The OC4J JSP container previously supported a standard XML-alternative syntax, but this is replaced with newer technology according to the current JSP specification.
  - The OC4J JSP container generates an XML view of every translated page, which is a mapping to an XML document that describes the page. This view is available for use by tag-library-validator classes.

  See Chapter 5, "JSP XML Support" for information about these features.

ORCL00444382

- Character encoding features:

  The OC4J JSP implementation supports the `pageEncoding` attribute of the `page` directive. This enables you to specify a character encoding for the page source that is different than the character encoding for the response (specified in the `contentType` attribute).

  See "Content Type Settings" on page 9-1.

Configurable JSP Extensions in OC4J

In addition to JSP 1.2 compliance, the OC4J JSP container in Oracle Application Server 10*g* Release 2 (10.1.2) includes the following notable features.

Also see "Overview of Oracle Value-Added Features" on page 2-9.

The following have been supported since the OC4J 9.0.3 implementation:

- Mode switch to avoid JSP translation errors if you have duplicate settings for the same directive attribute within a single JSP translation unit

  The JSP specification mandates translation errors if you have duplicate settings for the same directive attribute within a single JSP translation unit, except for the `page` directive `import` attribute. These errors may be unwanted or inappropriate, for example, if a page and an included file both set an attribute to the same value (such as `language="java"`).

  In "JSP Configuration Parameter Descriptions" on page 3-13, see the description of the `forgive_dup_dir_attr` parameter.

- Separate mode switches for XML validation of `web.xml` file and TLD files

  Validation of `web.xml` is disabled by default but can be enabled. Validation of TLD files is enabled by default but can be disabled.

  In "JSP Configuration Parameter Descriptions" on page 3-13, see the descriptions of the `xml_validate` and `no_tld_xml_validate` parameters.

- Mode flag for extra imports

  Use this to automatically import certain Java packages beyond the JSP defaults.

  In "JSP Configuration Parameter Descriptions" on page 3-13, see the description of the `extra_imports` parameter.

- "Well-known" location for sharing tag libraries

  You can specify a directory where tag library JAR files can be placed for sharing across multiple Web applications.

  In "JSP Configuration Parameter Descriptions" on page 3-13, see the description of the `well_known_taglib_loc` parameter.

- Configurable JSP timeout

  You can specify a timeout value for JSP pages, after which a page is removed from memory if it has not been requested again. In "JSP-Related OC4J Configuration Parameter Descriptions" on page 3-20, see the description of the `jsp-timeout` parameter.

The following features have been supported since the OC4J 9.0.2 implementation:

- Mode switch for automatic page retranslation and reloading

  You have a choice of: 1) running JSP pages without any automatic reloading or retranslation of JSP pages; 2) automatically reloading any page implementation

ORCL00444383

classes (but not JavaBeans or other dependency classes); or 3) automatically retranslating any JSP pages that have changed.

In "JSP Configuration Parameter Descriptions" on page 3-13, see the description of the `main_mode` parameter.

- Tag handler instance pooling

  To save time in tag handler creation and garbage collection, you can optionally enable pooling of tag handler instances. They are pooled in `application` scope. You can use different settings in different pages, or even in different sections of the same page. See "Disabling or Enabling Runtime or Compile-Time Tag Handler Reuse" on page 8-30.

- Output mode for null output

  You can print an empty string instead of the default "null" string for null output from a JSP page.

  In "JSP-Related OC4J Configuration Parameter Descriptions" on page 3-20, see the description of the `jsp-print-null` parameter.

- Single-threaded-model JSP instance pooling

  For single-threaded (non-thread-safe) JSP pages, page instances are pooled. There is no switch for this feature—it is always enabled.

Portability Across Servlet Environments

The JSP container is provided as a component of OC4J but is portable to other environments. Because the OC4J JSP container itself emulates certain required servlet features, this portability extends to older servlet environments. (Generally, a servlet 2.3 environment is required in order to support JSP 1.2 compliance.)

The servlet 2.0 specification was limited in that it provided only a single servlet context for each Java virtual machine, instead of a servlet context for each application. The OC4J JSP servlet emulation allows a full application framework in a servlet 2.0 environment, including providing applications with distinct servlet context and HTTP session objects.

Because of this extended functionality, the OC4J JSP container is not limited by the underlying servlet environment.

Oracle JDeveloper JSP Support

Visual Java programming tools now typically support JSP coding. In particular, Oracle JDeveloper supports JSP development and includes the following features:

- Integration of the OC4J JSP container to support the full application development cycle: editing, debugging, and running JSP pages
- Debugging of deployed JSP pages
- An extensive set of data-enabled and Web-enabled JavaBeans, known as JDeveloper Web beans
- The JSP Element Wizard, which offers a convenient way to add predefined Web beans to a page
- Support for incorporating custom JavaBeans
- A deployment option for JSP applications that rely on Oracle ADF Business Components, offered with JDeveloper

ORCL00444384

See "Application Deployment with Oracle JDeveloper" on page 7-25 for more information about JSP deployment support.

For debugging, JDeveloper can set breakpoints within JSP page source and can follow calls from JSP pages into JavaBeans. This is much more convenient than manual debugging techniques, such as adding print statements within the JSP page to output state into the response stream (for viewing in your browser) or to the server log (through the `log()` method of the implicit `application` object).

For information about JDeveloper, refer to the JDeveloper online help, or to the following site on the Oracle Technology Network:

```
http://www.oracle.com/technology/products/jdev/content.html
```

(You will need an Oracle Technology Network membership, which is free of charge.) For an overview of JSP tag libraries provided with JDeveloper, see the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Note: Other key IDE vendors have built plug-in modules that allow seamless integration with OC4J. This provides developers with the capability to build, deploy, and debug J2EE applications running on OC4J directly from within the IDE. You can refer to the following Web site for more information:

```
http://www.oracle.com/technology/products/ias/oracleas_
partners.html
```

Overview of Oracle Value-Added Features

OC4J value-added features for JSP pages can be grouped into three major categories:

- Features implemented through custom tag libraries, custom JavaBeans, or custom classes that are generally portable to other JSP environments
- Features that are Oracle-specific
- Features supporting caching technologies

The rest of this section provides feature summaries and overviews in these areas, plus a brief summary of Oracle support for the JavaServer Pages Standard Tag Library (JSTL). JSTL support is summarized more fully in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Tag Libraries and Utilities Provided with OC4J

This section lists extended OC4J JSP features that are implemented through standards-compliant custom tag libraries, custom JavaBeans, and other classes. These features are documented in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*:

- Extended types implemented as JavaBeans that can have a specified scope
- `JspScopeListener` for event-handling
- Integration with XML and XSL
- Data-access tag library (sometimes referred to as "SQL tags") and JavaBeans
- The JSP Markup Language (JML) custom tag library, which reduces the level of Java proficiency required for JSP development

- Personalization tags, for use with Oracle Application Server Personalization
- Web services tag library
- Tag libraries and JavaBeans for uploading files, downloading files, and sending e-mail from within an application
- EJB tag library
- Additional utility tags, such as for displaying dates and currency amounts appropriately for a specified locale

Important: The `JspScopeListener` and JML tag library are deprecated in the OC4J 10.1.2 implementation and will be desupported in future implementations.

Note: See "Overview of Tags and API for Caching Support" on page 2-11 for an overview of additional tag libraries for caching.

Overview of Oracle-Specific Features

This section provides an overview of Oracle-specific programming extensions supported by the OC4J JSP container:

- *Global includes*, a mechanism to automatically statically include a file or files in multiple pages
- Dynamic Monitoring Service (DMS) support for performance measurements
- Enhanced application framework and globalization support for servlet 2.0 environments

Global Includes

The OC4J JSP container provides a feature called *global includes*. You can use this feature to specify one or more files to statically include into JSP pages in or under a specified directory, through virtual JSP `include` directives. During translation, the JSP container looks for a configuration file, `/WEB-INF/ojsp-global-include.xml`, that specifies the included files and the directories for the pages.

This enhancement is particularly useful in migrating applications that had used `globals.jsa` or `translate_params` functionality in previous Oracle JSP releases. For more information, see "Oracle JSP Global Includes" on page 7-6.

Support for Dynamic Monitoring Service

DMS adds performance-monitoring features to a number of Oracle Application Server components, including OC4J. The goal of DMS is to provide information about runtime behavior through built-in performance measurements so that users can diagnose, analyze, and debug any performance problems. DMS provides this information in a package that can be used at any time, including during live deployment. Data are published through HTTP and can be viewed with a browser.

The OC4J JSP container supports DMS features, calculating relevant statistics and providing information to DMS servlets such as the spy servlet and monitoring agent. Statistics include the following (using averages, maximums, and minimums, as applicable). Times are in milliseconds.

ORCL00444386

- Processing time of HTTP request
- Processing time of JSP service method
- Number of JSP instances created or available
- Number of active JSP instances

(Counts of JSP instances are applicable only for single-threaded situations, where `isThreadSafe` is set to `false` in a `page` directive.)

Standard configuration for these servlets is in the OC4J `application.xml` and `default-web-site.xml` configuration file. Use the Enterprise Manager to access DMS, display DMS information, and, if appropriate, alter DMS configuration.

Also see the *Oracle Application Server Performance Guide*, which contains precise definitions of the JSP metrics and detailed instructions for viewing and analyzing them.

Enhanced Servlet 2.0 Support

OC4J supports special features for a servlet 2.0 environment. It is highly advisable to migrate to the OC4J servlet 2.3 environment as soon as practical, but in the meantime, be aware of the following:

- An enhanced application framework for servlet 2.0 environments
- Extended globalization support for servlet 2.0 environments

Overview of Tags and API for Caching Support

Faced with Web performance challenges, e-businesses must invest in more cost-effective technologies and services to improve the performance of their Internet sites. *Web caching*, the caching of both static and dynamic Web content, is a key technology in this area. Benefits of Web caching include performance, scalability, high availability, cost savings, and network traffic reduction.

OC4J provides the following support for Web caching technologies:

- The JESI tag library for Edge Side Includes (ESI), an XML-style markup language that allows dynamic content assembly away from the Web server

  The OracleAS Web Cache provides an ESI engine.

- A tag library and servlet API for the Web Object Cache, an application-level cache that is embedded and maintained within a Java Web application

  The Web Object Cache uses the Oracle Application Server Java Object Cache as its default repository.

These features are documented in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Support for the JavaServer Pages Standard Tag Library

With Oracle Application Server 10*g* Release 2 (10.1.2), the OC4J JSP product supports the JavaServer Pages Standard Tag Library (JSTL), as specified in the Sun Microsystems *JavaServer Pages Standard Tag Library, Version 1.0* specification.

JSTL is intended as a convenience for JSP page authors who are not familiar or not comfortable with scripting languages such as Java. Historically, scriptlets have been used in JSP pages to process dynamic data. With JSTL, the intent is for JSTL tag usage to replace the need for scriptlets.

Key JSTL features include the following:

- JSTL expression language (EL)

  The expression language further simplifies the code required to access and manipulate application data, making it possible to avoid request-time attributes as well as scriptlets.

- Core tags for expression language support, conditional logic and flow control, iterator actions, and access to URL-based resources
- Tags for XML processing, flow control, and XSLT transformations
- SQL tags for database access
- Tags for I18N-capable internationalization and formatting

  The term "I18N" refers to an internationalization standard.

Tag support is organized into four JSTL sublibraries according to these functional areas.

For a more complete summary of JSTL support, you can refer to the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*. For complete information about JSTL, refer to the specification at the following location:

`http://www.jcp.org/aboutJava/communityprocess/first/jsr052/index.html`

Note: The custom JML, XML, and data-access (SQL) tag libraries provided with OC4J pre-date JSTL and have areas of duplicate functionality. Going forward, for standards compliance, it is generally advisable to use JSTL instead of the custom libraries.

ORCL00444388

Exhibit 16

Oracle® Call Interface

Programmer's Guide,

11*g* Release 1 (11.1)

B28395-03

May 2008

Oracle Call Interface Programmer's Guide, 11*g* Release 1 (11.1)

B28395-03

Copyright © 1996, 2008, Oracle. All rights reserved.

Primary Author: Jack Melnick

Contributors: D. Alpern, Geeta Arora, Varun Arora, A. Bande, D. Banerjee, S. Banerjee, M. Bastawala, E. Belden, N. Bhatt, R. Chakravarthula, S. Chandrasekar, D. Chiba, L. Chidambaran, N. Ikeda, K. Itikarlapalli, Chandrasekhar Iyer, Shankar Iyer, B. Khaladkar, S. Kotsovolos, S. Krishnaswamy, S. Lari, Chao Liang, S. Lynn, A. Mullick, Valarie Moore, K. Neel, E. Paapanen, R. Pingte, R. Rajamani, M. Ramacher, A. Ramappa, A. Saxena, S. Seshadri, Rupa Singh, B. Sinha, H. Slattery, Mallikharjun Vemana, S. Vemuri, G. Viswanathan, S. Youssef

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

9

OCI Programming Advanced Topics

This chapter contains these topics:

- Connection Pooling in OCI
- Session Pooling in OCI
- Database Resident Connection Pooling
- When to Use Connection Pooling, Session Pooling, or Neither
- Statement Caching in OCI
- User-Defined Callback Functions in OCI
- Transparent Application Failover in OCI
- HA Event Notification
- OCI and Streams Advanced Queuing
- Publish-Subscribe Notification in OCI

Connection Pooling in OCI

Connection pooling is the use of a group (the pool) of reusable physical connections by several sessions, in order to balance loads. The management of the pool is done by OCI, not the application. Applications that can use connection pooling include middle-tier applications for Web application servers and e-mail servers.

A sample usage of this feature is in a Web application server connected to a back-end Oracle database. Suppose that a Web application server gets several concurrent requests for data from the database server. The application can create a pool (or a set of pools) in each environment during application initialization.

OCI Connection Pooling Concepts

Oracle has several transaction monitor capabilities such as the fine-grained management of database sessions and connections. This is done by separating the notion of database sessions (user handles) from connections (server handles). Using these OCI calls for session switching and session migration, it is possible for an application server or transaction monitor to multiplex several sessions over fewer physical connections, thus achieving a high degree of scalability by pooling of connections and back-end Oracle server processes.

The connection pool itself is normally configured with a shared pool of physical connections, translating to a back-end server pool containing an identical number of dedicated server processes.

The number of physical connections is less than the number of database sessions in use by the application. The number of physical connections and back-end server processes are also reduced by using connection pooling. Thus many more database sessions can be multiplexed.

Similarities and Differences from Shared Server

Connection pooling on the middle-tier is similar to what shared server offers on the back end. Connection pooling makes a dedicated server instance behave like a shared server instance by managing the session multiplexing logic on the middle tier.

The pooling of dedicated server processes including incoming connections into the dedicated server processes is controlled by the connection pool on the middle tier. The main difference between connection pooling and a shared server is that in the latter case, the connection from the client is normally to a dispatcher in the database instance. The dispatcher is responsible for directing the client request to an appropriate shared server. On the other hand, the physical connection from the connection pool is established directly from the middle-tier to the dedicated server process in the back-end server pool.

Connection pooling is beneficial only if the middle tier itself is multithreaded. Each thread can maintain a session to the database. The actual connections to the database are maintained by the connection pool and these connections (including the pool of dedicated database server processes) are shared among all the threads in the middle tier.

Stateless Sessions Versus Stateful Sessions

Stateless sessions are serially reusable across mid-tier threads. After a thread is done processing a database request on behalf of a three-tier user, the same database session can be reused for the purpose of a completely different request on behalf of a completely different three-tier user.

Stateful sessions to the database, on the other hand, are not serially reusable across mid-tier threads because they may have some particular state associated with a particular three-tier user. Examples of such state may be: open transactions, fetch state from a statement, PL/SQL package state, and so on. This makes the session not reusable for a different request for the duration that such state persists.

Note: Stateless sessions too may have open transactions, open statement fetch state, and so on. However, such a state persists for a relatively short duration (only during the processing of a particular three-tier request by a mid-tier thread) which allows the session to be serially reused for a different three-tier user (when such state is cleaned up).

Note: Stateless sessions are typically used in conjunction with Statement Caching.

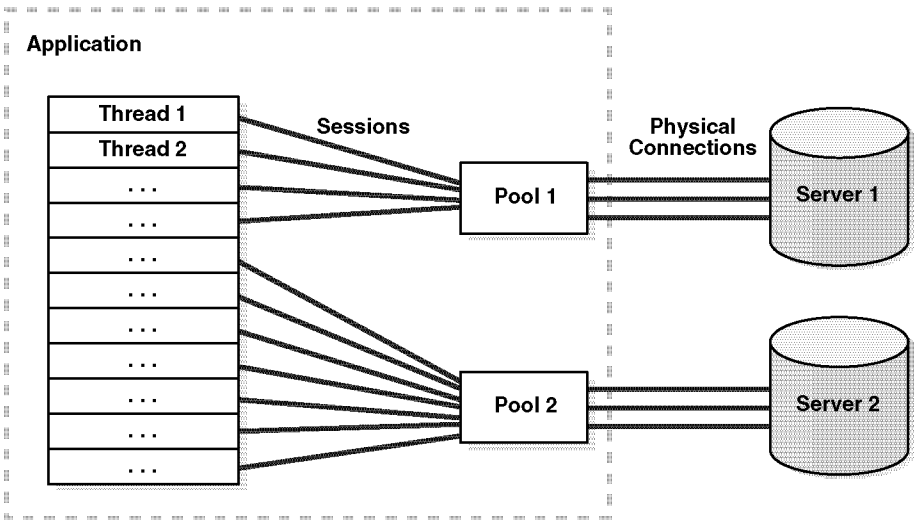
What connection pooling offers is stateless connections and stateful sessions. Users who need to work with stateless sessions, see "Session Pooling in OCI" on page 9-7.

Multiple Connection Pools

This advanced concept can be used for different database connections. Multiple connection pools can also be used when different priorities are assigned to users. Different service level guarantees can be implemented using connection pooling.

The following figure illustrates OCI connection pooling:

Figure 9–1 OCI Connection Pooling



Transparent Application Failover

Transaction Application Failover (TAF) is enabled for connection pooling. The concepts of TAF apply equally well with connections in the connection pool except that the `BACKUP` and `PRECONNECT` clauses should not be used in the connect string and do not work with connection pooling and TAF.

When a connection in the connection pool fails over, it uses the primary connect string itself to connect. Sessions failover when they use the pool for a database round trip after their instance failure. The listener would be configured to route it to a good instance if available, as is typical with service-based connect strings.

> **See Also:** *Oracle Database Net Services Administrator's Guide*, the chapter on Configuring Transparent Application Failover

OCI Calls for Connection Pooling

The steps in using connection pooling in your application are:

- Allocate the Pool Handle
- Create the Connection Pool
- Logon to the Database
- Deal with SGA Limitations in Connection Pooling
- Logoff from the Database
- Destroy the Connection Pool
- Free the Pool Handle

Allocate the Pool Handle

Connection pooling requires that the pool handle `OCI_HTYPE_CPOOL` be allocated by `OCIHandleAlloc()`. Multiple pools can be created for a given environment handle.

For a single connection pool, here is an allocation example:

```
OCICPool *poolhp;
OCIHandleAlloc((void *) envhp, (void **) &poolhp, OCI_HTYPE_CPOOL,
                      (size_t) 0, (void **) 0));
```

Create the Connection Pool

The function OCIConnectionPoolCreate() initializes the connection pool handle. It has these `IN` parameters:

- `connMin`, the minimum number of connections to be opened when the pool is created.
- `connIncr`, the incremental number of connections to be opened when all the connections are busy and a call needs a connection. This increment is used only when the total number of open connections is less than the maximum number of connections that can be opened in that pool.
- `connMax`, the maximum number of connections that can be opened in the pool. When the maximum number of connections are open in the pool, and all the connections are busy, if a call needs a connection, it will wait till it gets one. However, if the `OCI_ATTR_CONN_NOWAIT` attribute is set for the pool, an error is returned.
- A `poolUsername`, and a `poolPasswd`, to allow user sessions to transparently migrate between connections in the pool.
- In addition, an attribute `OCI_ATTR_CONN_TIMEOUT`, can be set to time out the connections in the pool. Connections idle for more than this time are terminated periodically, to maintain an optimum number of open connections. If this attribute is not set, then the connections are never timed out.

> **Note:** Shrinkage of the pool only occurs when there is a network round trip. If there are no operations, then the connections stay alive.

All the preceding attributes can be configured dynamically. So the application has the flexibility of reading the current load (number of open connections and number of busy connections) and tuning these attributes appropriately.

If the pool attributes (`connMax`, `connMin`, `connIncr`) are to be changed dynamically, `OCIConnectionPoolCreate()` must be called with `mode` set to `OCI_CPOOL_REINITIALIZE`.

The OUT parameters `poolName` and `poolNameLen` will contain values to be used in subsequent `OCIServerAttach()` and `OCILogon2()` calls in place of the database name and the database name length arguments.

There is no limit on the number of pools that can be created by an application. Middle tier applications can take advantage of this feature and create multiple pools to connect to the same server or to different servers, to balance the load based on the specific needs of the application.

Here is an example of this call:

```
OCIConnectionPoolCreate((OCIEnv *)envhp,
                  (OCIError *)errhp, (OCICPool *)poolhp,
                  &poolName, &poolNameLen,
                  (text *)database,strlen(database),
                  (ub4) conMin, (ub4) conMax, (ub4) conIncr,
                  (text *)pooluser,strlen(pooluser),
                  (text *)poolpasswd,strlen(poolpasswd),
                  OCI_DEFAULT));
```

Logon to the Database

The application will need to log on to the database for each thread, using one of the following interfaces.

- `OCILogon2()`

  This is the simplest interface. Use this interface when you need a simple Connection Pool connection and do not need to alter any attributes of the session handle. This interface can also be used to make proxy connections to the database.

  Here is an example using `OCILogon2()`:

```
for (i = 0; i < MAXTHREADS; ++i)
{
   OCILogon2(envhp, errhp, &svchp[i], "hr", 2, "hr", 2, poolName,
             poolNameLen, OCI_LOGON2_CPOOL));

}
```

  In order to use this interface to get a proxy connection, set the password parameter to `NULL`.

- `OCISessionGet()`

  This is the recommended interface. It gives the user the additional option of using external authentication methods, such as certificates, distinguished name, and so on. `OCISessionGet()` is the recommended uniform function call to retrieve a session.

  Here is an example using `OCISessionGet()`:

```
for (i = 0; i < MAXTHREADS; ++i)
{
        OCISessionGet(envhp, errhp, &svchp, authp,
                      (OraText *) poolName,
                      strlen(poolName), NULL, 0, NULL, NULL, NULL,
                      OCI_SESSGET_CPOOL)
 }
```

- `OCIServerAttach()` and `OCISessionBegin()`:

  Another interface can be used if the application needs to set any special attributes on the user session handle and server handle. For such a requirement, applications need to allocate all the handles (connection pool handle, server handles, session handles and service context handles).

  - Create the connection pool.
  - Call `OCIServerAttach()` with mode set to `OCI_CPOOL`.
  - Call `OCISessionBegin()` with mode set to `OCI_DEFAULT`.

The user should not set `OCI_MIGRATE` flag in the call to `OCISessionBegin()`, when the virtual server handle points to a connection pool (`OCIServerAttach()` called with `mode` set to `OCI_CPOOL`). Oracle supports passing this flag, `OCI_MIGRATE`, only for compatibility reasons. Do not use the `OCI_MIGRATE` flag, because the perception that the user gets when using a connection pool is of sessions having their own dedicated (virtual) connections which are transparently multiplexed onto real connections. Credentials can be set to `OCI_CRED_RDBMS` or `OCI_CRED_PROXY`. If the credentials are set to `OCI_CRED_PROXY`, only user name needs to be set on the session handle. (no explicit primary session needs to be created and `OCI_ATTR_MIGSESSION` need not be set).

Connection pooling does the multiplexing of a virtual server handle over physical connections transparently, hence eliminating the need for users to do so. The user gets the feeling of a session having a dedicated (virtual) connection. Since the multiplexing is done transparently to the user, users must not attempt to multiplex sessions over the virtual server handles themselves. The concepts of session migration and session switching, which require explicit multiplexing at the user level, are defunct in the case of connection pooling and should not be used.

In an OCI program, the user should create (`OCIServerAttach()` with mode set to `OCI_CPOOL`) a unique virtual server handle for each session that is created using the connection pool. There should be a one-to-one mapping between virtual server handles and sessions.

Deal with SGA Limitations in Connection Pooling

With `OCI_CPOOL` mode (connection pooling), the session memory (UGA) in the back-end database will come out of the SGA. This may require some SGA tuning on the back-end database to have a larger SGA if your application consumes more session memory than the SGA can accommodate. The memory tuning requirements for the back-end database will be similar to configuring the LARGE POOL in case of a shared server back end except that the instance is still in dedicated mode.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information, see the section on configuring Shared Server

If you are still running into the SGA limitation, you must consider:

- Reducing the session memory consumption by having fewer open statements for each session
- reducing the number of sessions in the back end by pooling sessions on the mid-tier or otherwise
- turning off connection pooling

The application must avoid using dedicated database links on the back end with connection pooling.

If the back end is a dedicated server, effective connection pooling will not be possible because sessions using dedicated database links will be tied to a physical connection rendering that same connection unusable by other sessions. If your application uses dedicated database links and you do not see effective sharing of back-end processes among your sessions, you must consider using shared database links.

> **See Also:** For more information about distributed databases, see the section on shared database links in *Oracle Database Administrator's Guide*

Logoff from the Database

Corresponding to the logon calls, these are the interfaces to use to log off from the database in connection pooling mode.

- `OCILogoff()`:

  If `OCILogon2()` was used to make the connection, `OCILogoff()` must be used to log off.

- `OCISessionRelease()`

  If `OCISessionGet()` was called to make the connection, then `OCISessionRelease()` must be called to log off.

- `OCISessionEnd()` and `OCIServerDetach()`

  If `OCIServerAttach()` and `OCISessionBegin()` were called to make the connection and start up the session, then `OCISessionEnd()` must be called to end the session and `OCIServerDetach()` must be called to release the connection.

Destroy the Connection Pool

Use OCIConnectionPoolDestroy() to destroy the connection pool.

Free the Pool Handle

The pool handle is freed using `OCIHandleFree()`.

These last three actions are illustrated in this code fragment:

```
for (i = 0; i < MAXTHREADS; ++i)
 {
   checkerr(errhp, OCILogoff((void *) svchp[i], errhp));
 }
 checkerr(errhp, OCIConnectionPoolDestroy(poolhp, errhp, OCI_DEFAULT));
 checkerr(errhp, OCIHandleFree((void *)poolhp, OCI_HTYPE_CPOOL));
```

See Also:

- "Connection Pool Handle Attributes" on page A-20
- "OCIConnectionPoolCreate()" on page 16-7, "OCILogon2()" on page 16-27, and "OCIConnectionPoolDestroy()" on page 16-9

Examples of OCI Connection Pooling

Examples of connection pooling in tested complete programs can be found in `cdemocp.c and cdemocpproxy.c` in directory `demo`.

Session Pooling in OCI

Session pooling means that the application will create and maintain a group of stateless sessions to the database. These sessions will be handed over to thin clients as requested. If no sessions are available, a new one may be created. When the client is done with the session, the client will release it to the pool. Thus, the number of sessions in the pool can increase dynamically.

Some of the sessions in the pool may be 'tagged' with certain properties. For instance, a user may request for a default session, set certain attributes on it, then label it or 'tag' it and return in to the pool. That user, or some other user, can require a session with the same attributes, and thus request for a session with the same tag. There may be several sessions in the pool with the same tag. The 'tag' on a session can be changed or reset.

See Also: "Using Tags in Session Pools" on page 9-8

Proxy sessions, too, can be created and maintained through this interface.

The behavior of the application when no free sessions are available and the pool has reached it's maximum size, will depend on certain attributes. A new session may be created or an error returned, or the thread may just block and wait for a session to become free.

The main benefit of this type of pooling will be performance. Making a connection to the database is a time-consuming activity, especially when the database is remote. Thus, instead of a client spending time connecting to the server, authenticating its credentials, and then receiving a valid session, it can just pick one from the pool.

Functionality of OCI Session Pooling

Session pooling has the following features:

- Create, maintain and manage a pool of stateless sessions transparently.
- Provide an interface for the application to create a pool and specify the minimum, increment and maximum number of sessions in the pool.
- Provide an interface for the user to obtain and release a default or 'tagged' session to the pool. A 'tagged' session is one with certain client-defined properties.
- Allow the application to dynamically change the number of minimum and maximum number of sessions.
- Provide a mechanism to always maintain an optimum number of open sessions, by closing sessions that have been idle for very long, and creating sessions when required.
- Allow for session pooling with authentication.

Homogeneous and Heterogeneous Session Pools

A session pool can be either homogeneous or heterogeneous. Homogeneous session pooling means that sessions in the pool are alike with respect to authentication (have the same user name and password and privileges). Heterogeneous session pooling means that you must provide authentication information because the sessions can have different security attributes and privileges.

Using Tags in Session Pools

The tags provide a way for users to customize sessions in the pool. A client may get a default or untagged session from a pool, set certain attributes on the session (such as NLS settings), and return the session to the pool, labeling it with an appropriate tag in the `OCISessionRelease()` call.

The user, or some other user, may request a session with the same tags in order to have a session with the same attributes, and can do so by providing the same tag in the `OCISessionGet()` call.

> **See Also:** "OCISessionGet()" on page 16-37 for a further discussion of tagging sessions.

OCI Handles for Session Pooling

Two handle types have been added for session pooling:

OCISPool

This is the session pool handle. It is allocated using `OCIHandleAlloc()`. It needs to be passed to `OCISessionPoolCreate()`, and `OCISessionPoolDestroy()`. It has the attribute type `OCI_HTYPE_SPOOL`.

An example of the `OCIHandleAlloc()` call follows:

```
OCISPool *spoolhp;
```

```
OCIHandleAlloc((void *) envhp, (void **) &spoolhp, OCI_HTYPE_SPOOL,
                        (size_t) 0, (void **) 0));
```

For an environment handle, multiple session pools can be created.

OCIAuthInfo

This is the authentication information handle. It is allocated using `OCIHandleAlloc()`. It is passed to `OCISessionGet()`. It supports all the attributes that are supported for user session handle. Please refer to user session handle attributes for more information. The authentication information handle has the attribute type `OCI_HTYPE_AUTHINFO`.

An example of the `OCIHandleAlloc()` call follows:

```
OCIAuthInfo *authp;
OCIHandleAlloc((void *) envhp, (void **) &authp, OCI_HTYPE_AUTHINFO,
                      (size_t) 0, (void **) 0));
```

See Also:

- "User Session Handle Attributes" on page A-13 for the attributes that belong to the authentication information handle.
- "Session Pool Handle Attributes" on page A-22 for more information about the session pooling attributes.
- "Connect, Authorize, and Initialize Functions" on page 16-3 for complete information about the functions used in session pooling.
- See "OCISessionGet()" on page 16-37 for details of the session handle attributes that can be used with this call.

Using OCI Session Pooling

The steps in writing a simple session pooling application which uses a user name and password are:

- Allocate the session pool handle using `OCIHandleAlloc()` for an `OCISPool` handle. Multiple session pools can be created for an environment handle.
- Create the session pool using OCISessionPoolCreate() with `mode` set to `OCI_DEFAULT` (for a new session pool). Refer to the function for a discussion of the other modes.
- Loop for each thread. Create the thread with a function that does the following:
- Allocate an authentication information handle of type `OCIAuthInfo` using `OCIHandleAlloc()`.
- Set the user name and password in the authentication information handle using `OCIAttrSet()`.
- Get a pooled session using OCISessionGet() with `mode` set to `OCI_SESSGET_SPOOL`.
- Perform the transaction.
- Commit or rollback the transactions.
- Release the session (logoff) with OCISessionRelease().
- Free the authentication information handle with `OCIHandleFree()`.

- End of the loop for each thread.
- Destroy the session pool using OCISessionPoolDestroy().

OCI Calls for Session Pooling

Here are the usages for OCI calls for session pooling.

Allocate the Pool Handle

Session pooling requires that the pool handle `OCI_HTYPE_SPOOL` be allocated by calling `OCIHandleAlloc()`.

Multiple pools can be created for a given environment handle. For a single session pool, here is an allocation example:

```
OCISPool *poolhp;
OCIHandleAlloc((void *) envhp, (void **) &poolhp, OCI_HTYPE_SPOOL, (size_t) 0,
               (void **) 0));
```

Create the Pool Session

The function `OCISessionPoolCreate()` can be used to create the session pool. Here is an example of how to use this call:

```
OCISessionPoolCreate(envhp, errhp, poolhp, (OraText **)&poolName,
              (ub4 *)&poolNameLen, database,
              (ub4)strlen((const signed char *)database),
              sessMin, sessMax, sessIncr,
              (OraText *)appusername,
              (ub4)strlen((const signed char *)appusername),
              (OraText *)apppassword,
              (ub4)strlen((const signed char *)apppassword),
              OCI_DEFAULT);
```

Logon to the Database

These are the interfaces that can be used to logon to the database in session pooling mode.

- `OCILogon2()`

  This is the simplest interface. However, it does not give the user the option of using tagging. Here is an example of how `OCILogon2()` can be used to log on to the database in session pooling mode:

  ```
  for (i = 0; i < MAXTHREADS; ++i)
  {
    OCILogon2(envhp, errhp, &svchp[i], "hr", 2, "hr", 2, poolName,
              poolNameLen, OCI_LOGON2_SPOOL));

  }
  ```

- `OCISessionGet()`

  This is the recommended interface. It gives the user the option of using tagging to label sessions in the pool, and thus make it easier to retrieve specific sessions. An example of using `OCISessionGet()` follows. It is taken from `cdemosp.c` in the `demo` directory.:

  ```
  OCISessionGet(envhp, errhp, &svchp, authInfop,
                (OraText *)database,strlen(database), tag,
                strlen(tag), &retTag, &retTagLen, &found,
  ```

```
                                    OCI_SESSGET_SPOOL);
```

Logoff from the Database

Corresponding to the preceding logon calls, these are the interfaces to use to log off from the database in session pooling mode.

- `OCILogoff()`

  If `OCILogon2()` was used to make the connection, `OCILogoff()` must be called to log off.

- `OCISessionRelease()`

  If `OCISessionGet()` was called to make the connection, then `OCISessionRelease()` must be called to log off. Pending transactions are automatically committed.

Destroy the Session Pool

`OCISessionPoolDestroy()` must be called to destroy the session pool. Here is an example of how this call can be made:

```
OCISessionPoolDestroy(poolhp, errhp, OCI_DEFAULT);
```

Free the Pool Handle

`OCIHandleFree()` must be called to free the session pool handle. Here is how this call can be made:

```
OCIHandleFree((void *)poolhp, OCI_HTYPE_SPOOL);
```

Note: Developers are advised to commit or rollback any open transaction before releasing the connection back to the pool. If it is not done, Oracle automatically commits any open transaction when the connection is released back to the pool.

Example of OCI Session Pooling

Here is an example of session pooling in a tested complete program:

See Also: `cdemosp.c` in directory `demo`.

Runtime Connection Load Balancing

Oracle Real Application Clusters (RAC) is a database option in which a single database is hosted by multiple instances on multiple nodes. RAC's shared disk method of clustering databases increases scalability because nodes can easily be added or freed to meet current needs and improve availability because if one node fails, another can assume its workload. It adds high availability and failover capacity to the database, because all instances have access to the whole database.

Balancing of work requests occurs at two different times: at connect time and at runtime. These are referred to as connect time load balancing provided by Oracle Net Services and runtime connection load balancing. For RAC environments, session pools use service metrics received from the RAC load balancing advisory through Fast Application Notification (FAN) events to balance application session requests. The work requests coming into the session pool can be distributed across the instances of RAC offering a service, using the current service performance.

See Also:

- `cdemosp.c` in the directory `demo`
- "Database Resident Connection Pooling" on page 9-12

Connect time load balancing occurs when a session is first created by the application. It is necessary that the sessions that are part of the pool be well distributed across RAC instances, at the time they are first created. This ensures that sessions on each of the instances get a chance to execute work.

Runtime connection load balancing is basically routing work requests to sessions in a session pool that best serve the work. It comes into effect when selecting a session from an existing session pool and thus is a very frequent activity. For session pools that support services at one instance only, the first available session in the pool is adequate. When the pool supports services that span multiple instances, there is a need to distribute the work requests across instances so that the instances that are providing better service or have greater capacity get more requests.

Runtime connection load balancing is enabled by default in a release 11.1 or higher client talking to a server of 10.2 or higher. Setting the `mode` parameter to `OCI_SPC_NO_RLB` when calling `OCISessionPoolCreate()` disables runtime connection load balancing.

Receiving Load Balancing Advisory FAN Events

To receive the service metrics based on the service time the following are required:

- RAC environment with Oracle Clusterware set up and enabled.
- The application must have been linked with the threads library.
- The OCI environment must be created in `OCI_EVENTS` and `OCI_THREADED` mode.
- The service should be modified to set up its goal and the connection load balancing goal:

```
EXEC DBMS_SERVICE.MODIFY_SERVICE("myService",
     DBMS_SERVICE.GOAL_SERVICE_TIME,
     clb_goal => dbms_service.clb_goal_short);
```

See Also:

- "OCISessionPoolCreate()" on page 16-41
- *Oracle Real Application Clusters Administration and Deployment Guide*, "Enabling OCI Clients to Receive FAN Events" section.
- *Oracle Database PL/SQL Packages and Types Reference*, "DBMS_SERVICE"

Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides a connection pool in the database server for typical Web application usage scenarios where the application acquires a database connection, works on it for a relatively short duration, and then releases it. DRCP pools "dedicated" servers, which is the equivalent of a server foreground and a database session combined and henceforth are referred to as the "pooled" servers. DRCP complements middle-tier connection pools that share connections between threads in a middle-tier process. In addition, DRCP enables sharing of database connections across middle-tier processes on the same middle-tier host and even across middle-tier hosts. This results in significant reduction in key

database resources needed to support a large number of client connections, thereby reducing the database tier memory footprint and boosting the scalability of both middle-tier and database tiers. Having a pool of readily available servers also has the additional benefit of reducing the cost of creating and tearing down client connections.

DRCP is especially relevant for architectures with multi-process single threaded application servers (such as PHP/Apache) that cannot do middle-tier connection pooling. The database can still scale to tens of thousands of simultaneous connections with DRCP.

The "pooled" server model closely follows the "dedicated" model that is used to connect to Oracle by default. It just removes the overhead of dedicating a server to every connection when the server is required only for executing short transactions. DRCP allows a connection to acquire and then voluntarily relinquish the "pooled" server, for use by other "similar" connections. On being acquired by a connection, a "pooled" server essentially transforms into a "dedicated" server for that connection, until it is released back into the pool. Clients getting connections out of the database resident connection pool connect to an Oracle background known as the connection broker. The connection broker implements the pool functionality and multiplexes pooled servers among inbound connections from the client processes.

See Also:

- *Oracle Database Concepts* for the details of the DRCP architecture
- *Oracle Database Administrator's Guide* for the details of configuring the database for DRCP

Session Purity and Connection Class

There are two new settings that can be specified when obtaining a session using `OCISessionGet()`.

Session Purity

Session purity specifies whether the application wants a "brand new" session or whether the application logic is set up to reuse a "pooled" session. `OCISessionGet()` has been enhanced to take in a purity setting of `OCI_SESSGET_PURITY_NEW` or `OCI_SESSGET_PURITY_SELF`. Alternatively, you can set `OCI_ATTR_PURITY_NEW` or `OCI_ATTR_PURITY_SELF` on the `OCIAuthInfo` handle prior to calling `OCISessionGet()`. Both these methods are equivalent.

Note: When reusing a session from the pool, the NLS attributes of the server take precedence over that of the client.

For example, if the client has `NLS_LANG` set to `french_france.us7ascii` and if it is assigned a German session from the pool, the client session would be German.

Connection classes can be used to restrict sharing and to avoid the above problem.

Setting Session Purity Example

A connection pooling application needs a `NEW` session.

```
/* OCIAttrSet method */

ub4 purity = OCI_ATTR_PURITY_NEW;
```

```
OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO,  &purity, sizeof (purity),
            OCI_ATTR_PURITY, errhp);
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
               NULL, NULL, NULL, OCI_SESSGET_SPOOL);
/* poolName is the name returned by OCISessionPoolCreate() */

/*  OCISessionGet mode method */
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
               NULL, NULL, NULL, OCI_SESSGET_SPOOL | OCI_SESSGET_PURITY_NEW);
/* poolName is the name returned by OCISessionPoolCreate() */
```

Connection Class

Connection class defines a logical name for the type of connection required by the application. Sessions from the OCISessionPool cannot be shared by different users - A session first created for user `HR` is only given out to subsequent requests by user `HR`. The connection class setting allows for further separation between the sessions of a given user. The connection class setting lets different applications (connecting as the same database user) identify their sessions using a logical name that corresponds to the application. OCI then ensures that such sessions belonging to a particular connection class are not shared outside of the connection class.

OCI provides a new attribute on the `OCIAuthInfo` handle, `OCI_ATTR_CONNECTION_CLASS` that can be used to set the connection class. The connection class is a string attribute. OCI supports a maximum connection class length of 1024 bytes. The character '*' is a special character and is not allowed in the connection class name.

Setting the Connection Class Examples

An HRMS application needs sessions identified with the connection class `HRMS`.

```
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "HRDB",
    strlen("HRDB"), 0, 10, 1, "HR", strlen("HR"), "HR", strlen("HR"),
    OCI_SPC_HOMOGENEOUS);

OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO, "HRMS", strlen ("HRMS"),
    OCI_ATTR_CONNECTION_CLASS, errhp);
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
    NULL, NULL, NULL, OCI_SESSGET_SPOOL);
```

A recruitment application needs sessions identified with the connection class `RECMS`.

```
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "HRDB",
    strlen("HRDB"), 0, 10, 1, "HR", strlen("HR"), "HR", strlen("HR"),
    OCI_SPC_HOMOGENEOUS);

OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO,  "RECMS", strlen("RECMS"),
    OCI_ATTR_CONNECTION_CLASS, errhp);
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
    NULL, NULL, NULL, OCI_SESSGET_SPOOL);
```

Defaults for Session Purity and Connection Class

Table 9–1, " Defaults Used in Various Client Scenarios" illustrates the defaults used in various client scenarios:

Table 9–1 Defaults Used in Various Client Scenarios

Attribute or Setting	Application Uses OCISessionGet() from Session Pool	Other Connections are not Obtained from OCISessionPool
OCI_ATTR_PURITY	OCI_ATTR_PURITY_SELF	OCI_ATTR_PURITY_NEW
OCI_ATTR_CONNECTION_CLASS	OCI-generated globally unique name for each client-side session pool that is used as the default connection class for all connections in the OCISessionPool.	SHARED
Sharing of sessions	Sharing of sessions between threads requesting sessions from the OCISessionPool.	Sharing among all connections of a particular database using the default SHARED connection class.

Starting the Database Resident Connection Pool

The database administrator (DBA) must logon as `SYSDBA` and start the default pool, `SYS_DEFAULT_CONNECTION_POOL`, using `DBMS_CONNECTION_POOL.START_POOL` with the default settings.

For detailed information about configuring the pool, see *Oracle Database Administrator's Guide*.

Enabling Database Resident Connection Pooling

Any application can leverage the database resident connection pool by specifying `:POOLED` in the EZ Connect string or `(SERVER=POOLED)` in the TNS connect string.

Example of TNS Connect String for Leveraging DRCP

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=oraclehost.company.com)
     (PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)(SERVER=POOLED)))
```

Example of EZ Connect String for Leveraging DRCP

```
oraclehost.company.com:1521/books.company.com:POOLED
```

Leveraging the Scalability of DRCP in an OCI Application

We consider three types of application scenarios and explain how each benefits from DRCP:

1. Applications that do not use OCISessionPool APIs and also do not specify any connection class/purity setting or specify a purity setting of `NEW` will simply get a brand new session from the DRCP. By the same token, when the application releases a connection back to the pool, the session is not shared with other instances of the same application by default. The application, however, does get the benefit of reusing an existing pooled server process.
2. Applications that use `OCISessionGet()` outside of an OCISessionPool, and specify the connection class and set `purity=SELF` will be able to reuse both DRCP pooled server processes and sessions. However, following an `OCISessionRelease()`, OCI will terminate the connection to the connection broker. On the next `OCISessionGet()` call, the application will reconnect to the broker. Once it reconnects, the DRCP will assign a pooled server (and session) belonging to the connection class specified. Reconnecting, however, incurs the cost of connection establishment and re-authentication. Such applications achieve better sharing of DRCP resources (processes and sessions) but do not get the benefit of caching connections to the connection broker.

3. Applications that use OCISessionPool APIs and specify the connection class and set `purity=SELF` make full use of the Database Resident Connection Pool functionality through reuse of both the pooled server process and the associated session and also getting the benefit of cached connections to the connection broker. Cached connections do not incur the cost of re-authentication on `OCISessionGet()`.

OCISessionPool APIs have been extended to interoperate with the database resident connection pool. Sessions obtained using `OCISessionGet()` from the OCISessionPool are obtained from the DRCP and sessions released to the OCISessionPool using `OCISessionRelease()` release the sessions to DRCP. The OCISessionPool also transparently keeps connections to the connection broker cached, for better performance.

Features offered by the traditional client-side OCISessionPool, such as tagging, statement caching and TAF (Transparent application failover) are supported with DRCP also.

For information on client-side session pooling APIs:

See Also:

- "OCISessionPoolCreate()" on page 16-41
- "OCISessionGet()" on page 16-37
- "OCISessionRelease()" on page 16-45
- "OCISessionPoolDestroy()" on page 16-44

Best Practices for Using DRCP

The steps to design an application that is able to leverage the full power of DRCP are very similar to the steps required for an application that uses the OCISessionPool.

The only additional step is that for best performance, when deployed to run with DRCP, it is recommended that the application specify an explicit connection class setting.

Multiple instances of the same application should specify the same connection class setting for best performance and enhanced sharing of DRCP resources. Take care to ensure that the different instances of the application can share database sessions.

Example of a Database Resident Connection Pooling Application

```
/* Assume all necessary handles are allocated */

/*   This middle-tier uses a single database user. Create a homogeneous
     client-side SP */
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "BOOKSDB",
    strlen("BOOKSDB"), 0, 10, 1, "SCOTT", strlen("SCOTT"), "password",
    strlen("password"), OCI_SPC_HOMOGENEOUS);

while (1)
{
   /* Process a client request */
   WaitForClientRequest();
   /* Application function */

   /* Set the Connection Class on the OCIAuthInfo handle that is passed as
      argument to OCISessionGet*/
```

```
   OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO,  "BOOKSTORE", strlen("BOOKSTORE"),
               OCI_ATTR_CONNECTION_CLASS, errhp);

   /* Purity need not be set as default is OCI_ATTR_PURITY_SELF for OCISessionPool
       connections */

   /* We can get a SCOTT session released by Mid-tier 2 */
   OCISessionGet(envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
                 NULL, NULL, NULL, OCI_SESSGET_SPOOL);

   /* Database calls using the svchp obtained above  */
   OCIStmtExecute(...)

   /* This releases the pooled server on the database for reuse */
   OCISessionRelease (svchp, errhp, NULL, 0, OCI_DEFAULT);
}

/* Mid-tier is done - exiting */
OCISessionPoolDestroy (spoolhp, errhp, OCI_DEFAULT);
```

Deployment Example 1

The code immediately preceding is deployed in 10 middle tier hosts servicing the BOOKSTORE application. The database used is 11*g* (or earlier) database in dedicated server mode but with DRCP not enabled. The client side has 11*g* libraries. The connect string used is:

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=oraclehost.company.com)
   (PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)))
```

In this case, the application gets dedicated server connections from the database.

Deployment Example 2

The code above is deployed in 10 middle tier hosts servicing the BOOKSTORE application. DRCP is enabled on the 11*g* database. Now all the middle tier processes can leverage the pooling capabilities offered by DRCP. The database resource requirement with DRCP will be much less than what would be required with dedicated server mode. Simply change the connect string to:

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=oraclehost.company.com)
  (PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)(SERVER=POOLED)))
```

Compatibility and Migration

An OCI application linked with 11*g* client libraries will work unaltered against a

- 11*g* database with DRCP disabled
- pre-11g database server
- 11*g* database server with DRCP enabled, when deployed with the DRCP connect string

Suitable clients benefit from enhanced scalability offered by DRCP provided that they are appropriately modified to use the OCISessionPool APIs with the connection class and purity settings as previously described.

Restrictions on Using Database Resident Connection Pooling

The following cannot be performed or used with pooled servers:

- Shut down the database.
- Stop the database resident connection pool.
- Change the password for the connected user.
- Use shared database links to connect to a database resident connection pool that is on a different instance.
- Use Advanced Security Options (ASO) such as encryption, certificates, and so on.
- Use migratable sessions on the server side, directly by using the `OCI_MIGRATE` option or indirectly using `OCIConnectionPool`.
- Use Initial Client Roles.
- Use Application Context attributes - `OCI_ATTR_APPCTX_NAME`, `OCI_ATTR_APPCTX_VALUE`, and so on.

When to Use Connection Pooling, Session Pooling, or Neither

If database sessions are not reusable by mid-tier threads (that is, they are stateful) and the number of back-end server processes may cause scaling problems on the database, use OCI connection pooling.

If database sessions are reusable by mid-tier threads (that is, they are stateless) and the number of back-end server processes may cause scaling problems on the database, use OCI session pooling.

If database sessions are not reusable by mid-tier threads (that is, they are stateful) and the number of back-end server processes will never be large enough to potentially cause any scaling issue on the database, there is no need to use any pooling mechanism.

> **Note:** Having non-pooled sessions/connections will result in tearing down and re-creation of the database session/connection for every mid-tier user request. This can cause severe scaling problems on the database side and excessive latency for the fulfillment of the request. Hence, it is strongly recommended that one of the pooling strategies be adopted for mid-tier applications based on whether the database session is stateful or stateless.

In connection pooling, the pool element is a connection and in session pooling, the pool element is a session.

As with any pool, the pooled resource is locked by the application thread for a certain duration until the thread has done its job on the database and the resource is released. The resource is unavailable to other threads during its period of use. Hence, application developers need to be aware that any kind of pooling works effectively with relatively short tasks. If the application is performing a long running transaction for example, it may deny the pooled resource to other sharers for long periods of time leading to starvation. Hence, pooling should be used in conjunction with short tasks and the size of the pool should be sufficiently large to maintain the desired concurrency of transactions.

Also, note that with:

1. OCI Connection Pool

   a. Connections to the database are pooled. Sessions are created and destroyed by the user. Each call to the database will pick up an appropriate available connection from the pool.

   b. The application is multiplexing several sessions over fewer physical connections to the database. The users can tune the pool configuration to achieve required concurrency.

   c. The life-time of the application sessions is independent of the life-time of the cached pooled connections.

2. OCI Session Pool

   Sessions and connections are pooled by OCI. The application gets sessions from the pool and releases sessions back to the pool.

Functions for Session Creation

The choices are:

1. `OCILogon()`

   This is the simplest way to get an OCI Session. The advantage is ease of obtaining an OCI service context. The disadvantage is that you cannot perform any advance OCI operations like session migration, proxy authentication, using a connection pool, or a session pool.

   See Also: "Application Initialization, Connection, and Session Creation" on page 2-15

2. `OCILogon2()`

   This includes the functionality of `OCILogon()` to get a session. This session may be a new one with a new underlying connection, or one that is started over a virtual connection from an existing connection pool, or one from an existing session pool. The `mode` parameter value that the function is called with determines its behavior.

   The user cannot modify the attributes (except `OCI_ATTR_STMTCACHESIZE`) of the service context returned by OCI.

   See Also: "OCILogon2()" on page 16-27

3. `OCISessionBegin()`

   This supports all the various options of an OCI session such as proxy authentication, getting a session from a connection pool or a session pool, external credentials, and migratable sessions. This is the lowest level call where all handles are needed to be explicitly allocated and all attributes set, and `OCIServerAttach()` is to be called prior to this call.

   See Also: "OCISessionBegin()" on page 16-33

4. `OCISessionGet()`

   This is now the recommended method to get a session. This session may be a new one with a new underlying connection, or one that is started over a virtual connection from an existing connection pool, or one from an existing session pool. The `mode` parameter value that the function is called with determines its behavior.

This works like `OCILogon2()` but additionally enables you to specify tags for obtaining specific sessions from the pool.

See Also: "OCISessionGet()" on page 16-37

Choosing Between Different Types of OCI Sessions

The choices are:

- Basic OCI Sessions

  This works by using user name and password over a dedicated OCI server handle. This is the no-pool mechanism. See earlier notes of when to use it.

  If authentication is obtained through external credentials, then user name or password is not required.

- Session Pool Sessions

  These sessions are from the session pool cache. Some sessions may be tagged. These are stateless sessions. Each `OCISessionGet()` and `OCISessionRelease()` call gets and releases a session from the session cache. This saves the server from creating and destroying sessions.

  See the earlier notes on connection pool sessions versus session pooling sessions versus no-pooling sessions.

- Connection Pool Sessions

  These are sessions created using `OCISessionGet()` and `OCISessionBegin()` calls from an OCI Connection Pool. There is no session cache as these are stateful sessions. Each call creates a new session and the user is responsible for terminating these sessions.

  The sessions are automatically migratable between the server handles of the connection pool. Each session can have user name and password or be a proxy session. See the earlier notes on connection pool sessions versus session pooling sessions versus no-pooling sessions.

- Sessions Sharing a Server Handle

  You can multiplex several OCI sessions over a few physical connections. The application does this manually by having the same server handle for these multiple sessions. It is preferred to have the session multiplexing details be left to OCI by using the OCI Connection Pool APIs.

- Proxy Sessions

  This is useful if the password of the client needs to be protected from the middle-tier. Proxy sessions can also be part of OCI Connection Pool and OCI Session Pool.

  See Also: "Middle-Tier Applications in OCI" on page 8-10

- Migratable Sessions

With transaction handles being migratable, there should be no need for applications to use this older feature, in light of OCI Connection Pooling.

See Also: "OCI Session Management" on page 8-9

Statement Caching in OCI

Statement caching refers to the feature that provides and manages a cache of statements for each session. In the server, it means that cursors are ready to be used without the need to parse the statement again. Statement caching can be used with connection pooling and with session pooling, and will improve performance and scalability. It can be used without session pooling as well. The OCI calls that implement statement caching are:

- `OCIStmtPrepare2()`
- `OCIStmtRelease()`

Statement Caching without Session Pooling in OCI

Users perform the usual OCI steps to logon. The call to obtain a session will have a mode that specifies whether statement caching is enabled for the session. Initially the statement cache will be empty. Developers will try to find a statement in the cache using the statement text. If the statement exists the API will return a previously prepared statement handle, otherwise it will return a newly prepared statement handle.

The application developer can perform binds and defines and then simply execute and fetch the statement before returning the statement back to the cache. In the latter case, where the statement handle was not found, the developer will need to set different attributes on the handle in addition to the other steps.

`OCIStmtPrepare2()` will also take a mode which will determine if the developer wants a prepared statement handle or a null statement handle if the statement is not found in the cache.

The pseudo code will look like:

```
OCISessionBegin( userhp, ... OCI_STMT_CACHE)  ;
OCIAttrset(svchp, userhp, ...);  /* Set the user handle in the service context */
OCIStmtPrepare2(svchp, &stmthp, stmttext, key, ...);
OCIBindByPos(stmthp, ...);
OCIDefineByPos(stmthp, ...);
OCIStmtExecute(svchp, stmthp, ...);
OCIStmtFetch(svchp, ...);
OCIStmtRelease(stmthp, ...);
...
```

Statement Caching with Session Pooling in OCI

The concepts remain the same, except that the statement cache is enabled at the session pool layer rather than at the session layer.

The attribute `OCI_ATTR_SPOOL_STMTCACHESIZE` sets the default statement cache size for each of the sessions in the session pool to this value. It is set on the `OCI_HTYPE_SPOOL` handle. The statement cache size for a particular session in the pool can be overridden at any time by using `OCI_ATTR_STMTCACHESIZE` on that session. The value of `OCI_ATTR_SPOOL_STMTCACHESIZE` can be changed at any time. This attribute can be used to enable or disable statement caching at the pool level, after creation, just as attribute `OCI_ATTR_STMTCACHESIZE` (on the service context) is used to enable or disable statement caching at the session level. This change will be reflected on individual sessions in the pool, when they are handed to a user. Tagged sessions are an exception to this behavior. This is explained later.

Enabling or disabling of statement caching is allowed on individual pooled sessions similar to non-pooled sessions.

A user can enable statement caching on a session retrieved from a non-statement cached pool in an `OCISessionGet()` or `OCILogon2()` call by specifying `OCI_SESSGET_STMTCACHE` or `OCI_LOGON2_STMTCACHE`, respectively, in the mode argument.

When a user asks for a session from a session pool, the statement cache size for that session will default to that of the pool. This may also mean enabling or disabling statement caching in that session. For example, if a pooled session (session A) has statement caching enabled, and statement caching is turned off in the pool, and a user asks for a session, and session A is returned, then statement caching will be turned off in Session A. As another example, if Session A in a pool does not have statement caching enabled, and statement caching at the pool level is turned on, then before returning session A to a user, statement caching on Session A with size equal to that of the pool is turned on.

This will not hold true if a tagged session is asked for and retrieved. In this case, the size of the statement cache will not be changed. Consequently, it will not be turned on or off. Moreover, if the user specifies mode `OCI_SESSGET_STMTCACHE` in the `OCISessionGet()` call, this will be ignored if the session is tagged. In our earlier example, if Session A was tagged, then it is returned as is to the user.

Rules for Statement Caching in OCI

Here are some rules to follow:

- Use the function `OCIStmtPrepare2()` instead of `OCIStmtPrepare()`. If you are using `OCIStmtPrepare()`, you are strongly urged not to use a statement handle across different service contexts. Doing so will raise an error if the statement has been obtained by `OCIStmtPrepare2()`. Migration of a statement handle to a new service context actually closes the cursor associated with the old session and therefore no sharing is achieved. Client-side sharing is also not obtained, because OCI will free all buffers associated with the old session when the statement handle is migrated.
- You are urged to keep one service context for each session and use statement handles only for that service context. That will be the preferred and recommended model and usage.
- A call to `OCIStmtPrepare2()`, even if the session does not have a statement cache, will also allocate the statement handle and therefore applications using only `OCIStmtPrepare2()` must not call `OCIHandleAlloc()` for the statement handle.
- A call to the `OCIStmtPrepare2()` must be followed with `OCIStmtRelease()` after the user is done with the statement handle. If statement caching is used, this will release the statement to the cache. If statement caching is not used, the statement will be deallocated. Do not call `OCIHandleFree()` to free the memory.
- If the call to `OCIStmtPrepare2()` is made with the `OCI_PREP2_CACHE_SEARCHONLY` mode and a `NULL` statement was returned (statement was not found), the subsequent call to `OCIStmtRelease()` is not required and must not be performed.
- Do not call `OCIStmtRelease()` for a statement that was prepared using `OCIStmtPrepare()`.

- The statement cache has a maximum size (number of statements) which can be modified by an attribute on the service context, `OCI_ATTR_STMTCACHESIZE`. The default value is 20. This attribute can also be used to enable or disable statement caching for the session, pooled or non-pooled. If `OCISessionBegin()` is called without mode set as `OCI_STMT_CACHE`, then `OCI_ATTR_STMTCACHESIZE` can be set on the service context to a nonzero attribute to turn on statement caching. If statement caching is not turned on at the session pool level, `OCISessionGet()` will return a non-statement cache-enabled session. `OCI_ATTR_STMTCACHESIZE` can be used to turn the caching on. Similarly the same attribute can be used to turn off statement caching by setting the cache size to zero.
- You can choose to tag a statement at the release time so that the next time you can request a statement of the same tag. The tag will be used to search the cache. An untagged statement (tag is `NULL`) is a special case of a tagged statement. Two statements are considered different if they only differ in their tags, or if one is untagged and the other is not.

  See Also: For information about the functions for statement caching,

  - "Statement Functions" on page 17-3
  - "Service Context Handle Attributes" on page A-8
  - "Session Pool Handle Attributes" on page A-22

OCI Statement Caching Code Example

Here is an example of statement caching:

See Also: Please refer to `cdemostc.c` in directory `demo` for a working example of statement caching.

User-Defined Callback Functions in OCI

The Oracle Call Interface has the capability to execute user-specific code in addition to OCI calls. This functionality can be used for:

- Adding tracing and performance measurement code to enable users to tune their applications.
- Performing pre- or post-processing code for specific OCI calls.
- Accessing other data sources with OCI by using the native OCI interface for Oracle databases and directing the OCI calls to use user callbacks for non-Oracle data sources.

The OCI callback feature has been added by providing support for calling user code before or after executing the OCI calls. Functionality has also been provided to allow the user-defined code to be executed instead of executing the OCI code.

The user callback code can also be registered dynamically without modifying the source code of the application. The dynamic registration is implemented by loading up to five user-created dynamically linked libraries after the initialization of the environment handle during the `OCIEnvCreate()` call. These user-created libraries (such as Dynamic Link Libraries (DLLs) on Windows, or shared libraries on Solaris, register the user callbacks for the selected OCI calls transparently to the application.

Sample Application

For a listing of the complete demonstration programs that illustrate the OCI user callback feature, see Appendix B, "OCI Demonstration Programs".

Registering User Callbacks in OCI

An application can register user callback libraries with the `OCIUserCallbackRegister()` function. Callbacks are registered in the context of the environment handle. An application can retrieve information about callbacks registered with a handle with the `OCIUserCallbackGet()` function.

> **See Also:** "OCIUserCallbackGet()" on page 17-190 and "OCIUserCallbackRegister()" on page 17-192

A user-defined callback is a subroutine that is registered against an OCI call and an environment handle. It can be specified to be either an entry callback, a replacement callback, or an exit callback.

- If it is an entry callback, it is called when the program enters the OCI function.
- Replacement callbacks are executed after entry callbacks. If the replacement callback returns a value of `OCI_CONTINUE`, then a subsequent replacement callback or the normal OCI-specific code is executed. If a replacement callback returns anything other than `OCI_CONTINUE`, subsequent replacement callbacks and the OCI code does not execute.
- After a replacement callback returns something other than `OCI_CONTINUE`, or an OCI function successfully executes, program control transfers to the exit callback (if one is registered).

If a replacement or exit callback returns anything other than `OCI_CONTINUE`, then the return code from the callback is returned from the associated OCI call.

A user callback can return `OCI_INVALID_HANDLE` when either an invalid handle or an invalid context is passed to it.

> **Note:** If any callback returns anything other than `OCI_CONTINUE`, then that return code is passed to the subsequent callbacks. If a replacement or exit callback returns a return code other than `OCI_CONTINUE`, then the final (not `OCI_CONTINUE`) return code is returned from the OCI call.

OCIUserCallbackRegister

A user callback is registered using the OCIUserCallbackRegister() call.

> **See Also:** "OCIUserCallbackRegister()" on page 17-192

Currently, `OCIUserCallbackRegister()` is only registered on the environment handle. The user's callback function of typedef `OCIUserCallback` is registered along with its context for the OCI call identified by the OCI function code, *fcode*. The type of the callback, whether entry, replacement, or exit, is specified by the *when* parameter.

For example, the `stmtprep_entry_dyncbk_fn` entry callback function and its context `dynamic_context`, are registered against the environment handle `hndlp` for the `OCIStmtPrepare()` call by calling the `OCIUserCallbackRegister()` function with the following parameters.

```
OCIUserCallbackRegister( hndlp,
                         OCI_HTYPE_ENV,
                         errh,
                         stmtprep_entry_dyncbk_fn,
                         dynamic_context,
                         OCI_FNCODE_STMTPREPARE,
                         OCI_UCBTYPE_ENTRY
                         (OCIUcb*) NULL);
```

User Callback Function

The user callback function has to follow the following syntax:

```
typedef sword (*OCIUserCallback)
     (void *ctxp,      /* context for the user callback*/
      void *hndlp,     /* handle for the callback, env handle for now */
      ub4 type,         /* type of handlp, OCI_HTYPE_ENV for this release */
      ub4 fcode,        /* function code of the OCI call */
      ub1 when,         /* type of the callback, entry or exit */
      sword returnCode, /* OCI return code */
      ub4 *errnop,      /* Oracle error number */
      va_list arglist); /* parameters of the oci call */
```

In addition to the parameters described in the `OCIUserCallbackRegister()` call, the callback is called with the return code, *errnop*, and all the parameters of the original OCI as declared by the prototype definition.

The return code is always passed in as `OCI_SUCCESS` and `*errnop` is always passed in as 0 for the first entry callback. Note that `*errnop` refers to the content of `errnop` because `errnop` is an IN/OUT parameter.

If the callback does not want to change the OCI return code, then it must return `OCI_CONTINUE`, and the value returned in `*errnop` is ignored. If on the other hand, the callback returns any other return code than `OCI_CONTINUE`, the last returned return code becomes the return code for the call. At the this point, the value of `*errnop` returned is set in the error handle, or in the environment handle if the error information is returned in the environment handle because of the absence of the error handle for certain OCI calls such as `OCIHandleAlloc()`.

For replacement callbacks, the `returnCode` is the non-`OCI_CONTINUE` return code from the previous callback or OCI call and `*errnop` is the value of the error number being returned in the error handle. This allows the subsequent callback to change the return code or error information if needed.

The processing of replacement callbacks is different in that if it returns anything other than `OCI_CONTINUE`, then subsequent replacement callbacks and OCI code is bypassed and processing jumps to the exit callbacks.

Note that if the replacement callbacks return `OCI_CONTINUE` to allow processing of OCI code, then the return code from entry callbacks is ignored.

All the original parameters of the OCI call are passed to the callback as variable parameters and the callback must retrieve them using the *va_arg* macros. The callback demonstration programs provide examples.

See Also: Appendix B, "OCI Demonstration Programs"

A null value can be registered to de-register a callback. That is, if the value of the callback (`OCIUserCallback()`) is `NULL` in the `OCIUserCallbackRegister()` call, then the user callback is de-registered.

When using the thread-safe mode, the OCI program acquires all mutexes before calling the user callbacks.

UserCallback Control Flow

This pseudocode describes the overall processing of a typical OCI call:

```
OCIXyzCall()
{
 Acquire mutexes on handles;
 retCode = OCI_SUCCESS;
 errno = 0;
 for all ENTRY callbacks do
  {

     EntryretCode = (*entryCallback)(..., retcode, &errno, ...);
     if (retCode != OCI_CONTINUE)
      {
         set errno in error handle or environment handle;
         retCode = EntryretCode;
       }
   }
  for all REPLACEMENT callbacks do
  {
   retCode = (*replacementCallback) (..., retcode, &errno, ...);
   if (retCode != OCI_CONTINUE)
      {
       set errno in error handle or environment handle
       goto executeEXITCallback;
       }
   }

   retCode = return code for XyzCall; /* normal processing of OCI call */

   errno = error number from error handle or env handle;

 executeExitCallback:
   for all EXIT callbacks do
   {
       exitRetCode = (*exitCallback)(..., retCode, &errno,...);
       if (exitRetCode != OCI_CONTINUE)
       {
           set errno in error handle or environment handle;
           retCode = exitRetCode;
       }
   }
    release mutexes;
    return retCode
}
```

UserCallback for OCIErrorGet()

If the callbacks are a total replacement of the OCI code, then they usually maintain their own error information in the call context and use that to return error information in `bufp` and `errnop` parameters of the replacement callback of the `OCIErrorGet()` call.

If on the other hand, the callbacks are either partially overriding OCI code, or just doing some other post processing, then they can use the exit callback to modify the error text and `errnop` parameters of the `OCIErrorGet()` by their own error message

and error number. Note that the `*errnop` passed into the exit callback is the error number in the error or the environment handle.

Errors from Entry Callbacks

If an entry callback wants to return an error to the caller of the OCI call, then it must register a replacement or exit callback. This is because if the OCI code is executed, then the error code from the entry callback is ignored. Therefore the entry callback must pass the error to the replacement or exit callback through its own context.

Dynamic Callback Registrations

Because user callbacks are expected to be used for monitoring OCI behavior or to access other data sources, it is desirable that the registration of the callbacks be done transparently and non-intrusively. This is accomplished by loading user-created dynamically linked libraries at OCI initialization time. These dynamically linked libraries are called *packages*. The user-created packages register the user callbacks for the selected OCI calls. These callbacks can further register or de-register user callbacks as needed when receiving control at runtime.

A makefile (`ociucb.mk` on Solaris) is provided with the OCI demonstration programs to create the package. The exact naming and location of this package is operating system dependent. The source code for the package must provide code for special callbacks that are called at OCI initialization and environment creation times.

The loading of the package is controlled by setting an operating system environment variable, `ORA_OCI_UCBPKG`. This variable names the packages in a generic way. The packages must be located in the `$ORACLE_HOME/lib` directory.

Loading Multiple Packages

The `ORA_OCI_UCBPKG` variable can contain a semicolon separated list of package names. The packages are loaded in the order they are specified in the list.

For example, previously one specified the package as:

```
setenv ORA_OCI_UCBPKG mypkg
```

Now, you can still specify the package as earlier, but in addition multiple packages can be specified as:

```
setenv ORA_OCI_UCBPKG "mypkg;yourpkg;oraclepkg;sunpkg;msoftpkg"
```

All these packages are loaded in order. That is, `mypkg` is loaded first and `msoftpkg` is loaded last.

A maximum of five packages can be specified.

Note: The sample makefile `ociucb.mk` creates `ociucb.so.1.0` on a Solaris or `ociucb.dll` on a Windows system. To load the `ociucb` package, the environmental variable `ORA_OCI_UCBPKG` must be set to `ociucb`. On Solaris, if the package name ends with `.so`, `OCIInitialize()` fails. The package name must end with `.so.1.0`.

For further details about creating the dynamic link libraries, read the Makefiles provided in the demo directory for your operating system. For further information on user-defined callbacks, see your operating system-specific documentation on compiling and linking applications.

Package Format

Previously a package had to specify the source code for the `OCIEnvCallback()` function. Now the `OCIEnvCallback()` function is obsolete. Instead, the package source must provide two functions. The first function has to be named as *packagename* suffixed with the word *Init*. For example, if the package is named `foo`, then the source file (for example, but not necessarily, foo.c) must contain a `fooInit()` function with a call to `OCISharedLibInit()` function specified exactly as:

```
sword fooInit(metaCtx, libCtx, argfmt, argc, argv)
      void *     metaCtx;         /* The metacontext */
      void *     libCtx;          /* The context for this package. */
      ub4        argfmt;          /* package argument format */
      sword      argc;            /* package arg count*/
      void *     argv[];          /* package arguments */
{
  return  (OCISharedLibInit(metaCtx, libCtx, argfmt, argc, argv,
                            fooEnvCallback));
}
```

The last parameter of the `OCISharedLibInit()` function, `fooEnvCallback()`, in this case, is the name of the second function. It can be named anything, but by convention it can be named *packagename* suffixed with the word *EnvCallback*.

This function is a replacement for `OCIEnvCallback()`. Now all the dynamic user callbacks must be registered in this function. The function must be of type `OCIEnvCallbackType`, which is specified as:

```
typedef sword (*OCIEnvCallbackType)(OCIEnv *env, ub4 mode,
                                    size_t xtramem_sz, void *usrmemp,
                                    OCIUcb *ucbDesc);
```

When an environment handle is created, then this callback function is called at the very end. The `env` parameter is the newly created environment handle.

The `mode`, `xtramem_sz`, and `usrmemp` are the parameters passed to the `OCIEnvCreate()` call. The last parameter, `ucbDesc`, is a descriptor that is passed to the package. The package uses this descriptor to register the user callbacks as described later.

A sample `ociucb.c` file is provided in the demo directory. The makefile `ociucb.mk` is also provided (on Solaris) in the `demo` directory to create the package. Please note that this may be different on other operating systems. The demo directory also contains full user callback demo programs (`cdemoucb.c, cdemoucbl.c`,) illustrating this.

User Callback Chaining

User callbacks can both be registered statically in the application itself or dynamically at runtime in the DLLs. A mechanism is needed to allow the application to override a previously registered callback and then later invoke the overridden one in the newly registered callback to preserve the behavior intended by the dynamic registrations. This can result in chaining of user callbacks.

For this purpose, the `OCIUserCallbackGet()` function is provided to find out which function and context is registered for an OCI call.

See Also: "OCIUserCallbackGet()" on page 17-190

Accessing Other Data Sources Through OCI

Because Oracle is the predominant database accessed, applications can take advantage of the OCI interface to access non-Oracle data by using the user callbacks to access them. This allows an application written in OCI to access Oracle data without any performance penalty. To access non-Oracle data sources, drivers can be written that access the non-Oracle data in user callbacks. Because OCI provides a very rich interface, there is usually a straightforward mapping of OCI calls to most data sources. This solution is better than writing applications for other middle layers such as ODBC that introduce performance penalties for all data sources. Using OCI does not incur any penalty for the common case of accessing Oracle data sources, and incurs the same penalty that ODBC does for non-Oracle data sources.

Restrictions on Callback Functions

There are certain restrictions on the usage of callback functions, including `OCIEnvCallback()`:

- A callback cannot call other OCI functions except `OCIUserCallbackRegister()`, `OCIUserCallbackGet()`, `OCIHandleAlloc()`, `OCIHandleFree()`. Even for these functions, if they are called in a user callback, then callbacks on them are not called to avoid recursion. For example, if `OCIHandleFree()` is called in the callback for `OCILogoff()`, then the callback for `OCIHandleFree()` is disabled during the execution of the callback for `OCILogoff()`.
- A callback cannot modify OCI data structures such as the environment or error handles.
- A callback cannot be registered for `OCIUserCallbackRegister()` call itself, or for any of the following:
  - `OCIUserCallbackGet()`
  - `OCIEnvCreate()`
  - `OCIInitialize()`
  - `OCIEnvInit()`

Example of OCI Callbacks

For example, lets suppose that there are five packages each registering entry, replacement, and exit callbacks for the `OCIStmtPrepare()` call. That is, the `ORA_OCI_UCBPKG` variable is set as:

```
setenv ORA_OCI_UCBPKG "pkg1;pkg2;pkg3;pkg4;pkg5"
```

In each package pkgN (where N can be 1 through 5), the pkgNInit() and PkgNEnvCallback() functions are specified as:

```
pkgNInit(void *metaCtx, void *libCtx, ub4 argfmt, sword argc, void **argv)
{
  return OCISharedLibInit(metaCtx, libCtx, argfmt, argc, argv, pkgNEnvCallback);
}
```

The pkgNEnvCallback() function registers the entry, replacement, and exit callbacks as:

```
pkgNEnvCallback(OCIEnv *env, ub4 mode, size_t xtramemsz,
                                 void *usrmemp, OCIUcb *ucbDesc)
{
  OCIHandleAlloc((void *)env, (void **)&errh, OCI_HTYPE_ERROR, (size_t) 0,
        (void **)NULL);

  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, pkgN_entry_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_ENTRY, ucbDesc);

  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, pkgN_replace_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_REPLACE, ucbDesc);

  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, pkgN_exit_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_EXIT, ucbDesc);

  return OCI_CONTINUE;
}
```

Finally, in the source code for the application, user callbacks can be registered with the NULL *ucbDesc* as:

```
  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, static_entry_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_ENTRY, (OCIUcb *)NULL);

  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, static_replace_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_REPLACE, (OCIUcb *)NULL);

  OCIUserCallbackRegister(env, OCI_HTYPE_ENV, errh, static_exit_callback_fn,
        pkgNctx, OCI_FNCODE_STMTPREPARE, OCI_UCBTYPE_EXIT, (OCIUcb *)NULL);
```

When the OCIStmtPrepare() call is executed, the callbacks are called in the following order:

```
static_entry_callback_fn()
pkg1_entry_callback_fn()
pkg2_entry_callback_fn()
pkg3_entry_callback_fn()
pkg4_entry_callback_fn()
pkg5_entry_callback_fn()

static_replace_callback_fn()
 pkg1_replace_callback_fn()
  pkg2_replace_callback_fn()
   pkg3_replace_callback_fn()
    pkg4_replace_callback_fn()
     pkg5_replace_callback_fn()

      OCI code for OCIStmtPrepare call

pkg5_exit_callback_fn()
```

```
pkg4_exit_callback_fn()
pkg3_exit_callback_fn()
pkg2_exit_callback_fn()
pkg1_exit_callback_fn()

static_exit_callback_fn()
```

Note: The exit callbacks are called in the reverse order of the entry and replacement callbacks

The entry and exit callbacks can return any return code and the processing continues to the next callback. However, if the replacement callback returns anything other than `OCI_CONTINUE`, then the next callback (or OCI code if it is the last replacement callback) in the chain is bypassed and processing jumps to the exit callback. For example, if `pkg3_replace_callback_fn()` returned `OCI_SUCCESS`, then `pkg4_replace_callback_fn()`, `pkg5_replace_callback_fn()`, and the OCI processing for the `OCIStmtPrepare()` call is bypassed. Instead, `pkg5_exit_callback_fn()` is executed next.

OCI Callbacks from External Procedures

There are several OCI functions that can be used as callbacks from external procedures.

See Also: These functions are listed in Chapter 20, "OCI Cartridge Functions". For information about writing C subroutines that can be called from PL/SQL code, including a list of which OCI calls can be used, and some example code, refer to the *Oracle Database Advanced Application Developer's Guide*

Transparent Application Failover in OCI

Transparent Application Failover (TAF) is a client-side feature designed to minimize disruptions to end-user applications that occur when database connectivity fails because of instance or network failure. TAF can be implemented on a variety of system configurations including Real Application Clusters and Data Guard Physical Standby Databases. TAF can also be used after re-starting a single instance system, for example when repairs are made.

TAF can be configured to restore database sessions and, optionally, replay open queries. In previous versions, TAF with the `SELECT` failover option would be engaged only on the statement that was in use at the time of a failure. For example, if there were 10 statement handles in use by the application, and statement 7 was the failure-time statement (the statement in use when the failure happened), statements 1-6 and 8-10 would have to be re-executed after statement 7 was failed over using TAF.

Starting with 10g Release 2 (10.2), this has been improved. Now all statements that an application attempts to use after a failure will attempt failover. That is, an attempt to execute or fetch against other statements will engage TAF recovery just as for the failure-time statement. Notably, this means that subsequent statements may now succeed (whereas in the past they failed), or the application may receive errors corresponding to an attempted TAF recovery (like ORA-25401).

Note: TAF is not supported for remote database links or for DML statements.

Configuring Transparent Application Failover

TAF can be configured on both the client-side and the server-side. If both are configured, server-side settings take precedence.

Configure TAF on the client side by including the `FAILOVER_MODE` parameter in the `CONNECT_DATA` portion of a connect descriptor.

> **See Also:** *Oracle Database Net Services Reference* for more information about client-side configuration of TAF (Connect Data Section).

Configure TAF on the server-side by modifying the target service with the `DBMS_SERVICE.MODIFY_SERVICE` packaged procedure.

An initial attempt at failover may not always be successful. The OCI provides a mechanism for retrying failover after an unsuccessful attempt.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information on the server-side configuration of TAF (DBMS_SERVICE).

Transparent Application Failover Callbacks in OCI

Because of the delay that can occur during failover, the application developer may want to inform the user that failover is in progress, and request that the user wait for notification that failover is complete. Additionally, the session on the initial instance may have received some `ALTER SESSION` commands. These `ALTER SESSION` commands will not be automatically replayed on the second instance. Consequently, the developer may wish to replay them on the second instance.

To accommodate these requirements, the application developer can register a failover callback function. In the event of failover, the callback function is invoked several times during the course of reestablishing the user's session.

The first call to the callback function occurs when the database first detects an instance connection loss. This callback is intended to allow the application to inform the user of an upcoming delay. If failover is successful, a second call to the callback function occurs when the connection is reestablished and usable.

Once the connection has been re-established, the client may wish to replay `ALTER SESSION` commands and inform the user that failover has happened. If failover is unsuccessful, then the callback is called to inform the application that failover will not take place. Additionally, the callback is called each time a user handle besides the primary handle is re-authenticated on the new connection. Since each user handle represents a server-side session, the client may wish to replay `ALTER SESSION` commands for that session.

> **See Also:** See "Handling OCI_FO_ERROR" on page 9-35 for more information about this scenario

Failover Callback Structure and Parameters

The basic structure of a user-defined application failover callback function is as follows:

```
sb4 appfocallback_fn ( void       * svchp,
                       void       * envhp,
                       void       * fo_ctx,
                       ub4        fo_type,
                       ub4        fo_event );
```

An example is provided in the section "Failover Callback Example" on page 9-34 for the following parameters:

svchp
The first parameter, `svchp`, is the service context handle. It is of type `void *`.

envhp
The second parameter, `envhp`, is the OCI environment handle. It is of type `void *`.

fo_ctx
The third parameter, `fo_ctx`, is a client context. It is a pointer to memory specified by the client. In this area the client can keep any necessary state or context. It is passed as a `void *`.

fo_type
The fourth parameter, `fo_type`, is the failover type. This lets the callback know what type of failover the client has requested. The usual values are:

- `OCI_FO_SESSION`, which indicates that the user has requested only session failover.
- `OCI_FO_SELECT`, which indicates that the user has requested select failover as well.

fo_event
The last parameter is the failover event. This indicates to the callback why it is being called. It has several possible values:

- `OCI_FO_BEGIN` indicates that failover has detected a lost connection and failover is starting.
- `OCI_FO_END` indicates successful completion of failover.
- `OCI_FO_ABORT` indicates that failover was unsuccessful, and there is no option of retrying.
- `OCI_FO_ERROR` also indicates that failover was unsuccessful, but it gives the application the opportunity to handle the error and retry failover.
- `OCI_FO_REAUTH` indicates that you have multiple authentication handles and failover has occurred after the original authentication. It indicates that a user handle has been re-authenticated. To find out which, the application checks the `OCI_ATTR_SESSION` attribute of the service context handle (which is the first parameter).

Failover Callback Registration

For the failover callback to be used, it must be registered on the server context handle. This registration is done by creating a callback definition structure and setting the `OCI_ATTR_FOCBK` attribute of the server handle to this structure.

The callback definition structure must be of type `OCIFocbkStruct`. It has two fields: `callback_function`, which contains the address of the function to call, and `fo_ctx` which contains the address of the client context.

An example of callback registration is included as part of the example in the next section.

Failover Callback Example

The following code shows an example of a simple user-defined callback function definition, registration, and unregistration.

Part 1: Failover Callback Definition

```
sb4  callback_fn(svchp, envhp, fo_ctx, fo_type, fo_event)
void * svchp;
void * envhp;
void *fo_ctx;
ub4 fo_type;
ub4 fo_event;
{
switch (fo_event)
   {
   case OCI_FO_BEGIN:
   {
     printf(" Failing Over ... Please stand by \n");
     printf(" Failover type was found to be %s \n",
                    ((fo_type==OCI_FO_SESSION) ? "SESSION"
                    :(fo_type==OCI_FO_SELECT) ? "SELECT"
                    : "UNKNOWN!"));
     printf(" Failover Context is :%s\n",
                   (fo_ctx?(char *)fo_ctx:"NULL POINTER!"));
     break;
   }
   case OCI_FO_ABORT:
   {
     printf(" Failover stopped. Failover will not take place.\n");
     break;
   }
   case    OCI_FO_END:
   {
       printf(" Failover ended ...resuming services\n");
     break;
   }
   case OCI_FO_REAUTH:
   {
       printf(" Failed over user. Resuming services\n");
     break;
   }
   default:
   {
     printf("Bad Failover Event: %d.\n",  fo_event);
     break;
   }
   }
   return 0;
}
```

Part 2: Failover Callback Registration

```
int register_callback(srvh, errh)
void *srvh; /* the server handle */
OCIError *errh; /* the error handle */
{
  OCIFocbkStruct failover;                /*  failover callback structure */
  /* allocate memory for context */
  if (!(failover.fo_ctx = (void *)malloc(strlen("my context.")+1)))
     return(1);
```

```
  /* initialize the context. */
  strcpy((char *)failover.fo_ctx, "my context.");
  failover.callback_function = &callback_fn;
  /* do the registration */
  if (OCIAttrSet(srvh, (ub4) OCI_HTYPE_SERVER,
                 (void *) &failover, (ub4) 0,
                 (ub4) OCI_ATTR_FOCBK, errh)  != OCI_SUCCESS)
     return(2);
  /* successful conclusion */
  return (0);
}
```

Part 3: Failover Callback Unregistration

```
OCIFocbkStruct failover;   /*  failover callback structure */
sword status;

  /* set the failover context to null */
  failover.fo_ctx = NULL;
  /* set the failover callback to null */
  failover.callback_function = NULL;
  /* un-register the callback */
  status = OCIAttrSet(srvhp, (ub4) OCI_HTYPE_SERVER,
                      (void *) &failover, (ub4) 0,
                      (ub4) OCI_ATTR_FOCBK, errhp);
```

Handling OCI_FO_ERROR

A failover attempt is not always successful. If the attempt fails, the callback function receives a value of `OCI_FO_ABORT` or `OCI_FO_ERROR` in the `fo_event` parameter. A value of `OCI_FO_ABORT` indicates that failover was unsuccessful, and no further failover attempts are possible. `OCI_FO_ERROR`, on the other hand, provides the callback function with the opportunity to handle the error in some way. For example, the callback may choose to wait a specified period of time and then indicate to the OCI library that it must reattempt failover.

Note: This functionality is only available to applications linked with the 8.0.5 or later OCI libraries running against any Oracle server.

Failover does not work if a LOB column is part of the select list.

Consider the following time line of events:

Table 9–2 Time and Event

Time	Event
T0	Database fails (failure lasts until T5).
T1	Failover triggered by user activity.
T2	User attempts to reconnect; attempt fails.
T3	Failover callback invoked with `OCI_FO_ERROR`.
T4	Failover callback enters predetermined sleep period.
T5	Database comes back up again.
T6	Failover callback triggers new failover attempt; it is successful.

Table 9–2 (Cont.) Time and Event

Time	Event
T7	User successfully reconnects

The callback function triggers the new failover attempt by returning a value of `OCI_FO_RETRY` from the function.

The following example code shows a callback function which can be used to implement the failover strategy similar to the scenario described earlier. In this case the failover callback enters a loop in which it sleeps and then reattempts failover until it is successful:

```
/*---------------------------------------------------------------------*/
/* the user defined failover callback  */
/*---------------------------------------------------------------------*/
sb4  callback_fn(svchp, envhp, fo_ctx, fo_type, fo_event )
void * svchp;
void * envhp;
void *fo_ctx;
ub4 fo_type;
ub4 fo_event;
{
   OCIError *errhp;
   OCIHandleAlloc(envhp, (void **)&errhp, (ub4) OCI_HTYPE_ERROR,
              (size_t) 0, (void **) 0);
   switch (fo_event)
   {
   case OCI_FO_BEGIN:
   {
     printf(" Failing Over ... Please stand by \n");
     printf(" Failover type was found to be %s \n",
            ((fo_type==OCI_FO_NONE) ? "NONE"
             :(fo_type==OCI_FO_SESSION) ? "SESSION"
             :(fo_type==OCI_FO_SELECT) ? "SELECT"
             :(fo_type==OCI_FO_TXNAL) ? "TRANSACTION"
             : "UNKNOWN!"));
     printf(" Failover Context is :%s\n",
            (fo_ctx?(char *)fo_ctx:"NULL POINTER!"));
     break;
   }
   case OCI_FO_ABORT:
   {
     printf(" Failover aborted. Failover will not take place.\n");
     break;
   }
   case    OCI_FO_END:
   {
       printf("\n Failover ended ...resuming services\n");
     break;
   }
   case OCI_FO_REAUTH:
   {
       printf(" Failed over user. Resuming services\n");
     break;
   }
   case OCI_FO_ERROR:
   {
     /* all invocations of this can only generate one line. The newline
      * will be put at fo_end time.
```

```
      */
      printf(" Failover error gotten. Sleeping...");
      sleep(3);
      printf("Retrying. ");
      return (OCI_FO_RETRY);
      break;
    }
    default:
    {
      printf("Bad Failover Event: %d.\n",  fo_event);
      break;
    }
    }
    return 0;
}
```

HA Event Notification

Suppose an user employs a Web browser to log in to an application server that accesses a back-end database server. Failure of the database instance can result in a wait that can be up to in minutes in duration before the failure is known to the user. The ability to quickly detect failures of server instances, communicate this to the client, close connections, and clean up idle connections in connection pools is provided by HA event notification.

For High Availability clients connected to a Real Application Clusters (RAC) database, HA event notification can be used to provide a best-effort programmatic signal to the client in the event of a database failure. Client applications can register a callback on the environment handle to signal interest in this information. When a significant failure event occurs (which applies to a connection made by this client), the callback is invoked, with information concerning the event (the event payload), and a list of connections (server handles) that were disconnected as a result of the failure.

For example, consider a client application that has two connections to instance A and two connections to instance B of the same database. If instance A goes down, a notification of the event will be sent to the client, which will then disconnect the two connections to instance B, and invoke the registered callback. Note that if another instance, C, of the same database, goes down, the client will not be notified (since it will not affect any of the client's connections).

The HA event notification mechanism improves the response time of the application in the presence of failure. In the past, a failure would result in the connection being broken only after the TCP time out expired, which could take minutes. With HA event notification, standalone, connection pool, and session pool connections are automatically broken and cleaned up by OCI and the application callback is invoked within seconds of the failure event. If any of these server handles are TAF-enabled, failover will also automatically be engaged by OCI.

Applications must connect to a RAC instance to enable HA event notification. Furthermore, these applications must:

- Initialize the OCI Environment in `OCI_EVENTS` mode.
- Connect to a service that has notifications enabled (use the `DBMS_SERVICE.MODIFY_SERVICE` procedure to set `AQ_HA_NOTIFICATIONS` to `TRUE`).
- Link with a thread library.

Then these applications can register a callback that is invoked whenever an HA event occurs.

OCIEvent Handle

The `OCIEvent` handle encapsulates the attributes from the event payload. OCI implicitly allocates this handle prior to calling the event callback, which can obtain the read-only attributes of the event by calling `OCIAttrGet()`. Memory associated with these attributes is only valid for the duration of the event callback.

See Also: "Event Handle Attributes" on page A-73

OCI Failover for Connection and Session Pools

A connection pool in an instance of Real Application Clusters (RAC) consists of a pool of connections connected to different instances of the RAC. Upon receiving the node failure notification, all the connections connected to that particular instance should be cleaned up. For the connections that are in use, OCI has to close the connections: transparent application failover (TAF) occurs immediately and those connections will be reestablished. The connections that are idle and in the free list of the pool have to be purged, so that a bad connection is never returned back to the user from the pool.

To accommodate custom connection pools, OCI will provide a callback function that can be registered on the environment handle. If registered, this callback is invoked when an HA event occurs. Sessions pools are treated the same way as connection pools. Note that server handles from OCI connection pools or session pools will not be passed to the callback. Hence in some cases, the callback could be called with an empty list of connections.

OCI Failover for Independent Connections

No special handling is required for independent connections; all such connections that are connected to failed instances are immediately disconnected. For idle connections, TAF will be engaged to reestablish the connection when the connection is used on a subsequent OCI call. Connections that are in use at the time of the failure event will be broken out immediately, so that TAF can begin. Note that this applies for the "in-use" connections of connection and session pools also.

Event Callback

The event callback, of type `OCIEventCallback`, has the following signature:

```
void evtcallback_fn (void     *evtctx,
                     OCIEvent  *eventhp );
```

where `evtctx` is the client context and `OCIEvent` is an event handle which is opaque to the OCI library. The other input argument is `eventhp`, the event handle: that is, the attributes associated with an event.

If registered, this function will be called once for each event. In the case of RAC HA events, this callback will be invoked after the affected connections have been disconnected. The following environment handle attributes are used to register an event callback and context, respectively:

- `OCI_ATTR_EVTCBK` is of datatype `OCIEventCallback *`. It is read-only.
- `OCI_ATTR_EVTCTX` is of datatype `void *`. It is also read-only.

```
text *myctx = "dummy context"; /* dummy context passed to callback fn */
```

```
...
/* OCI_ATTR_EVTCBK and OCI_ATTR_EVTCTX are read-only. */
OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV, (void *) evtcallback_fn,
           (ub4) 0, (ub4) OCI_ATTR_EVTCBK, errhp);
OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV, (void *) myctx,
           (ub4) 0, (ub4) OCI_ATTR_EVTCTX, errhp);
...
```

Within the OCI Event callback, the list of affected server handles is encapsulated in the `OCIEvent` handle. For RAC HA DOWN events, client applications can iterate over a list of server handles that are affected by the event by using `OCIAttrGet()` with attribute types `OCI_ATTR_HA_SVRFIRST` and `OCI_ATTR_HA_SVRNEXT`:

```
OCIAttrGet(eventhp, OCI_HTYPE_EVENT, (void *)&srvhp, (ub4 *)0,
           OCI_ATTR_HA_SRVFIRST, errhp);
/* or, */
OCIAttrGet(eventhp, OCI_HTYPE_EVENT, (void *)&srvhp, (ub4 *)0,
           OCI_ATTR_HA_SRVNEXT, errhp);
```

When called with attribute `OCI_ATTR_HA_SRVFIRST`, this function will retrieve the first server handle in the list of server handles affected. When called with attribute `OCI_ATTR_HA_SRVNEXT`, this function will retrieve the next server handle in the list. This function will return `OCI_NO_DATA` and `srvhp` will be a `NULL` pointer, when there are no more server handles to return.

`srvhp` is an output pointer to a server handle whose connection has been closed as a result of an HA event. `errhp` is an error handle to populate. The application returns an `OCI_NO_DATA` error when there are no more affected server handles to retrieve.

When retrieving the list of server handles that have been affected by an HA event, be aware that the connection has already been closed and many server handle attributes are no longer valid. Instead, use the user memory segment of the server handle to store any per-connection attributes required by the event notification callback. This memory remains valid until the server handle is freed.

Custom Pooling: Tagged Server Handles

The following features apply to custom pools:

1. You can tag a server handle with its parent connection object if it is created on behalf of a custom pool. Use the "user memory" parameters of `OCIHandleAlloc()` to request that the server handle be allocated with a user memory segment. A pointer to the "user memory" segment is returned by `OCIHandleAlloc()`.
2. When an HA event occurs, and an affected server handle has been retrieved, there is a means to retrieve the server handle's tag information so appropriate cleanup can be performed. The attribute `OCI_ATTR_USER_MEMORY` is used to retrieve a pointer to a handle's user memory segment. `OCI_ATTR_USER_MEMORY` is valid for all user-allocated handles. If the handle was allocated with extra memory, this attribute will return a pointer to the user memory. A `NULL` pointer will be returned for those handles not allocated with extra memory. This attribute is read-only and is of datatype `void*`.

Note: You are free to define the precise contents of the server handle's user memory segment to facilitate cleanup activities from within the HA event callback (or for other purposes if needed) because OCI does not write or read from this memory in any way. The user memory segment is freed along with the `OCIHandleFree()` call on the server handle.

Event Notification Example

```
sword retval;
OCIServer *srvhp;
struct myctx {
   void *parentConn_myctx;
   uword numval_myctx;
};
typedef struct myctx myctx;
myctx  *myctxp;
/* Allocate a server handle with user memory - pre 10.2 functionality */
if (retval = OCIHandleAlloc(envhp, (void **)&srvhp, OCI_HTYPE_SERVER,
                            (size_t)sizeof(myctx), (void **)&myctxp)
/* handle error */
myctxp->parentConn_myctx = <parent connection reference>;

/* In an event callback function, retrieve the pointer to the user memory */
evtcallback_fn(void *evtctx, OCIEvent *eventhp)
{
  myctx *ctxp = (myctx *)evtctx;
  OCIServer *srvhp;
  OCIError *errhp;
  sb4       retcode;
  retcode = OCIAttrGet(eventhp, OCI_HTYPE_SERVER, &srvhp, (ub4 *)0,
                       OCI_ATTR_HA_SRVFIRST, errhp);
  while (!retcode) /* OCIAttrGet will return OCI_NO_DATA if no more srvhp */
  {
     OCIAttrGet((void *)srvhp, OCI_HTYPE_SERVER, (void *)&ctxp,
                (ub4)0, (ub4)OCI_ATTR_USER_MEMORY, errhp);
           /* Remove the server handle from the parent connection object */
     retcode = OCIAttrGet(eventhp, OCI_HTYPE_SERVER, &srvhp, (ub4 *)0,
                          OCI_ATTR_HA_SRVNEXT, errhp);
...
  }
...
}
```

Determining Transparent Application Failover (TAF) Capabilities

You can have the application adjust its behavior if a connection is or is not TAF-enabled. Use `OCIAttrGet()` as follows to find out if a server handle is TAF-enabled:

```
boolean taf_capable;
...
OCIAttrGet(srvhp, (ub4) OCI_HTYPE_SERVER, (void *) &taf_capable,
           (ub4) sizeof(taf_capable), (ub4)OCI_ATTR_TAF_ENABLED, errhp);
...
```

where `taf_capable` is a boolean variable, which this call sets to `TRUE` if the server handle is TAF-enabled, and `FALSE` if not; `srvhp` is an input target server handle;

OCI_ATTR_TAF_ENABLED is a an attribute which is a pointer to a boolean variable and is read-only; errhp is an input error handle.

OCI and Streams Advanced Queuing

The OCI provides an interface to Streams Advanced Queuing (Streams AQ) feature. Streams AQ provides message queuing as an integrated part of the Oracle server. Streams AQ provides this functionality by integrating the queuing system with the database, thereby creating a *message-enabled database*. By providing an integrated solution Streams AQ frees application developers to devote their efforts to their specific business logic rather than having to construct a messaging infrastructure.

Note: In order to use Streams Advanced Queuing, you must be using the Enterprise Edition

See Also:

- *Oracle Streams Advanced Queuing User's Guide*
- *Oracle XML Developer's Kit Programmer's Guide*
- For example code demonstrating the use of the OCI with AQ, refer to the description of OCIAQEnq() on page 17-106

OCI Streams Advanced Queuing Functions

The OCI library includes several functions related to Streams Advanced Queuing:

- OCIAQEnq()
- OCIAQDeq()
- OCIAQListen()
- OCIAQListen2()
- OCIAQEnqArray()
- OCIAQDeqArray()

You can enqueue an array of messages to a single queue. The messages all share the same enqueue options, but each message in the array can have different message properties. You can also dequeue an array of messages from a single queue. For transaction group queues, you can dequeue all messages for a single transaction group using one call.

See Also: "Streams Advanced Queuing and Publish-Subscribe Functions" on page 17-101

OCI Streams Advanced Queuing Descriptors

The following descriptors are used by OCI Streams AQ operations:

- OCIAQEnqOptions
- OCIAQDeqOptions
- OCIAQMsgProperties

- OCIAQAgent

You can allocate these descriptors with respect to the service handle using the standard OCIDescriptorAlloc() call. The following code shows examples of this:

```
OCIDescriptorAlloc(svch, &enqueue_options, OCI_DTYPE_AQENQ_OPTIONS, 0, 0 );
OCIDescriptorAlloc(svch, &dequeue_options, OCI_DTYPE_AQDEQ_OPTIONS, 0, 0 );
OCIDescriptorAlloc(svch, &message_properties, OCI_DTYPE_AQMSG_PROPERTIES, 0, 0);
OCIDescriptorAlloc(svch, &agent, OCI_DTYPE_AQAGENT, 0, 0 );
```

Each descriptor has a variety of attributes which can be set or read.

> **See Also:** "Streams Advanced Queuing Descriptor Attributes" on page A-38

Streams Advanced Queuing in OCI Versus PL/SQL

The following tables compare functions, parameters, and options for OCI Streams AQ functions and descriptors, and PL/SQL AQ functions in the DBMS_AQ package.

Table 9–3 AQ Functions

PL/SQL Function	OCI Function
DBMS_AQ.ENQUEUE	OCIAQEnq()
DBMS_AQ.DEQUEUE	OCIAQDeq()
DBMS_AQ.LISTEN	OCIAQListen(), OCIAQListen2()
DBMS_AQ.ENQUEUE_ARRAY	OCIAQEnqArray()
DBMS_AQ.DEQUEUE_ARRAY	OCIAQDeqArray()

The following table compares the parameters for the enqueue functions:

Table 9–4 Enqueue Parameters

DBMS_AQ.ENQUEUE Parameter	OCIAQEnq() Parameter
queue_name	queue_name
enqueue_options	enqueue_options
message_properties	message_properties
payload	payload
msgid	msgid
-	Note: OCIAQEnq() also requires the following additional parameters: svch, errh, payload_tdo, payload_ind, and flags.

The following table compares the parameters for the dequeue functions:

Table 9–5 Dequeue Parameters

DBMS_AQ.DEQUEUE Parameter	OCIAQDeq() Parameter
queue_name	queue_name
dequeue_options	dequeue_options
message_properties	message_properties

Table 9–5 (Cont.) Dequeue Parameters

DBMS_AQ.DEQUEUE Parameter	OCIAQDeq() Parameter
payload	payload
msgid	msgid
-	Note: OCIAQDeq() also requires the following additional parameters: svch, errh, dequeue_options, message_properties, payload_tdo, payload, payload_ind, and flags.

The following table compares parameters for the listen functions:

Table 9–6 Listen Parameters

DBMS_AQ.LISTEN Parameter	OCIAQListen2() Parameter
agent_list	agent_list
wait	wait
agent	agent
listen_delivery_mode	lopts
-	Note: OCIAQListen2() also requires the following additional parameters: svchp, errhp, agent_list, num_agents, agent, lmops, and flags.

The following table compares parameters for the array enqueue functions:

Table 9–7 Array Enqueue Parameters

DBMS_AQ.ENQUEUE_ARRAY Parameter	OCIAQEnqArray() Parameter
queue_name	queue_name
enqueue_options	enqopt
array_size	iters
message_properties_array	msgprop
payload_array	payload
msgid_array	msgid
-	Note: OCIAQEnqArray() also requires the following additional parameters: svch, errh, payload_tdo, payload_ind, ctxp, enqcbfp, and flags.

The following table compares parameters for the array dequeue functions:

Table 9–8 Array Dequeue Parameters

DBMS_AQ.DEQUEUE_ARRAY Parameter	OCIAQDeqArray() Parameter
queue_name	queue_name
dequeue_options	deqopt

Table 9–8 (Cont.) Array Dequeue Parameters

DBMS_AQ.DEQUEUE_ARRAY Parameter	OCIAQDeqArray() Parameter
array_size	iters
message_properties_array	msgprop
payload_array	payload
msgid_array	msgid
-	Note: OCIAQDeqArray() also requires the following additional parameters: svch, errh, msgprop, payload_tdo, payload_ind, ctxp, deqcbfp, and flags.

The following table compares parameters for the agent attributes:

Table 9–9 Agent Parameters

PL/SQL Agent Parameter	OCIAQAgent Attribute
name	OCI_ATTR_AGENT_NAME
address	OCI_ATTR_AGENT_ADDRESS
protocol	OCI_ATTR_AGENT_PROTOCOL

The following table compares parameters for the message properties:

Table 9–10 Message Properties

PL/SQL Message Property	OCIAQMsgProperties Attribute
priority	OCI_ATTR_PRIORITY
delay	OCI_ATTR_DELAY
expiration	OCI_ATTR_EXPIRATION
correlation	OCI_ATTR_CORRELATION
attempts	OCI_ATTR_ATTEMPTS
recipient_list	OCI_ATTR_RECIPIENT_LIST
exception_queue	OCI_ATTR_EXCEPTION_QUEUE
enqueue_time	OCI_ATTR_ENQ_TIME
state	OCI_ATTR_MSG_STATE
sender_id	OCI_ATTR_SENDER_ID
transaction_group	OCI_ATTR_TRANSACTION_NO
original_msgid	OCI_ATTR_ORIGINAL_MSGID
delivery_mode	OCI_ATTR_MSG_DELIVERY_MODE

The following table compares enqueue option attributes:

Table 9–11 Enqueue Option Attributes

PL/SQL Enqueue Option	OCIAQEnqOptions Attribute
visibility	OCI_ATTR_VISIBILITY

Table 9–11 (Cont.) Enqueue Option Attributes

PL/SQL Enqueue Option	OCIAQEnqOptions Attribute
relative_msgid	OCI_ATTR_RELATIVE_MSGID
sequence_deviation	OCI_ATTR_SEQUENCE_DEVIATION (deprecated)

The following table compares dequeue option attributes:

Table 9–12 Dequeue Option Attributes

PL/SQL Dequeue Option	OCIAQDeqOptions Attribute
consumer_name	OCI_ATTR_CONSUMER_NAME
dequeue_mode	OCI_ATTR_DEQ_MODE
navigation	OCI_ATTR_NAVIGATION
visibility	OCI_ATTR_VISIBILITY
wait	OCI_ATTR_WAIT
msgid	OCI_ATTR_DEQ_MSGID
correlation	OCI_ATTR_CORRELATION
delivery_mode	OCI_ATTR_MSG_DELIVERY_MODE

Note: OCIAQEnq() returns the error ORA-25219 while specifying the enqueue option OCI_ATTR_SEQUENCE along with OCI_ATTR_RELATIVE_MSGID. This happens when enqueuing two messages. For the second message, enqueue options OCI_ATTR_SEQUENCE and OCI_ATTR_RELATIVE_MSGID is set to dequeue this message before the first one. It does not return an error if you do not specify the 'sequence' but, of course, the message is not dequeued before the relative message.

OCIAQEnq() does not return an error if OCI_ATTR_SEQUENCE attribute is not set, but the message is not dequeued before the message with relative message Id.

Buffered Messaging

Buffered messaging is a non-persistent messaging capability within Streams AQ, that was first available in Oracle Database 10*g* Release 2. Buffered messages reside in shared memory, and can be lost in the event of instance failure. Unlike persistent messages, no redo is written to disk. Buffered message enqueue and dequeue is much faster than persistent message operations. Since shared memory is limited, buffered messages may have to be spilled to disk. Flow control may be enabled to prevent applications from flooding the shared memory when the message consumers are slow or have stopped for some reason. The following functions are used for buffered messaging:

See Also:

- "OCIAQEnq()" on page 17-106
- "OCIAQDeq()" on page 17-102
- "OCIAQListen2()" on page 17-111

Enqueue Buffered Messaging Example

```
...
OCIAQMsgProperties  *msgprop;
OCIAQEnqueueOptions *enqopt;
message              msg;    /* message is an object type */
null_message         nmsg;   /* message indicator */
...
/* Allocate descriptors */
  OCIDescriptorAlloc(envhp, (void **)&enqopt, OCI_DTYPE_AQENQ_OPTIONS, 0,
                     (void **)0));

 OCIDescriptorAlloc(envhp, (void **)&msgprop,OCI_DTYPE_AQMSG_PROPERTIES, 0,
                    (void **)0));

/* Set delivery mode to buffered */
 dlvm = OCI_MSG_BUFFERED;
 OCIAttrSet(enqopt,  OCI_DTYPE_AQENQ_OPTIONS, (void *)&dlvm, sizeof(ub2),
            OCI_ATTR_MSG_DELIVERY_MODE, errhp);
/* Set visibility to Immediate (visibility must always be immediate for buffered
   messages) */
vis = OCI_ENQ_ON_COMMIT;

OCIAttrSet(enqopt, OCI_DTYPE_AQENQ_OPTIONS,(void *)&vis, sizeof(ub4),
           OCI_ATTR_VISIBILITY, errhp)

/* Message was an object type created earlier, msg_tdo is its Type
   Descriptor Object */
OCIAQEnq(svchp, errhp, "Test_Queue", enqopt, msgprop, msg_tdo, (void **)&mesg,
         (void **)&nmesg, (OCIRaw **)0, 0));
...
```

Dequeue Buffered Messaging Example

```
...
OCIAQMsgProperties  *msgprop;
OCIAQDequeueOptions *deqopt;
...
OCIDescriptorAlloc(envhp, (void **)&mprop, OCI_DTYPE_AQMSG_PROPERTIES, 0,
                   (void **)0));
OCIDescriptorAlloc(envhp, (void **)&deqopt, OCI_DTYPE_AQDEQ_OPTIONS, 0,
                   (void **)0);

/* Set visibility to Immediate (visibility must always be immediate for buffered
   message operations) */
vis = OCI_ENQ_ON_COMMIT;
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,(void *)&vis, sizeof(ub4),
           OCI_ATTR_VISIBILITY, errhp)
/* delivery mode is buffered */
dlvm  = OCI_MSG_BUFFERED;
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (void *)&dlvm,  sizeof(ub2),
           OCI_ATTR_MSG_DELIVERY_MODE, errhp);
/* set the consumer for which to dequeue the message (this needs to be specified
   regardless of the type of message being dequeued.
```

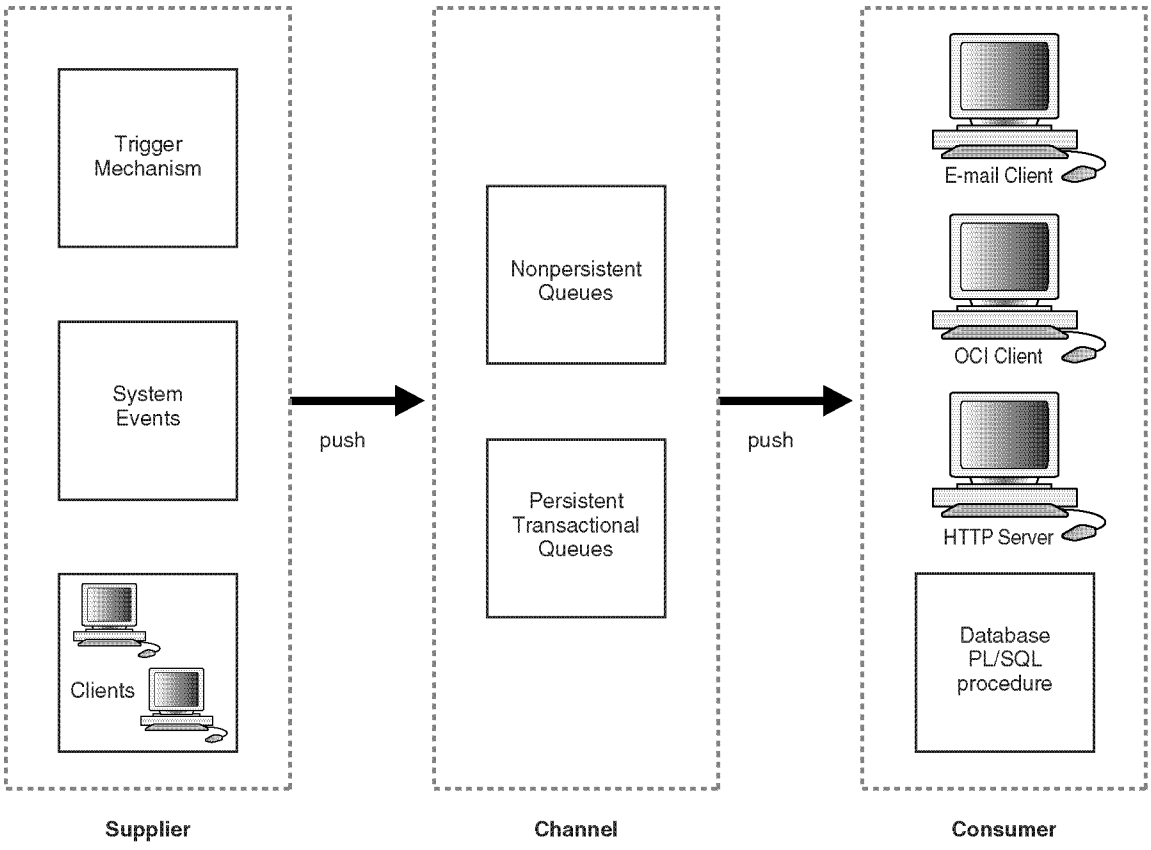
```
*/
consumer = "FIRST_SUBSCRIBER";
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (void *)consumer,
           (ub4)strlen((char*)consumer), OCI_ATTR_CONSUMER_NAME, errhp);
/* Dequeue the message but don't return the payload (to simplify the code
   snippet)
*/
OCIAQDeq(svchp, errhp,  "test_queue", deqopt, msgprop, msg_tdo, (void **)0,
         (void **)0, (OCIRaw**)0, 0);
...
```

Note: Array operations are not supported for buffered messaging. Applications can use the `OCIAQEnqArray()` and `OCIAQDeqArray()` functions with array size set to 1.

Publish-Subscribe Notification in OCI

The publish-subscribe notification feature allows an OCI application to receive client notifications directly, register an e-mail address to which notifications can be sent, register a HTTP URL to which notifications can be posted, or register a PL/SQL procedure to be invoked on a notification. Figure 9–2, "Publish-Subscribe Model" illustrates the process.

Figure 9–2 Publish-Subscribe Model



An OCI application can:

- register interest in notifications in the AQ namespace and be notified when an enqueue occurs.
- register interest in subscriptions to database events and receive notifications when the events are triggered.
- manage registrations, such as disabling registrations temporarily or dropping the registrations entirely.
- post, or send, notifications to registered clients.

In all the preceding scenarios the notification can be received directly by the OCI application, or the notification can be sent to a pre-specified e-mail address, or it can be sent to a pre-defined HTTP URL, or a pre-specified database PL/SQL procedure can be invoked as a result of a notification.

Registered clients are notified asynchronously when events are triggered or on an explicit AQ enqueue. Clients do not need to be connected to a database.

See Also:

- For information on Streams Advanced Queuing, see "OCI and Streams Advanced Queuing" on page 9-41
- For information on creating queues and about Streams AQ, including concepts, features, and examples, refer to *Oracle Streams Advanced Queuing User's Guide*
- For information on creating triggers, refer to the chapter on `CREATE TRIGGER` in the *Oracle Database SQL Language Reference*.

Publish-Subscribe Registration Functions in OCI

Registration can be done in two ways:

- You register directly to the database. This way is simple and the registration will take effect immediately.
- Open Registration. You register using LDAP, from which the database receives the registration request. This is useful when the client cannot have a database connection (the client wants to register for a database open event while the database is down), or if the client wants to register for the same event or events in multiple databases at one time.

Let us next consider these two alternative ways of registration.

Publish-Subscribe Register Directly to the Database

The following steps are required in an OCI application to register and receive notifications for events. It is assumed that the appropriate event trigger or AQ queue has been set up. The initialization parameter `COMPATIBLE` must be set to 8.1 or higher.

See Also:

- Detailed descriptions of the functions noted can be found in "Streams Advanced Queuing and Publish-Subscribe Functions" on page 17-101
- For examples of the use of these functions in an application, see "Publish-Subscribe Direct Registration Example" on page 9-57

Note: The publish-subscribe feature is only available on multithreaded operating systems.

1. Call `OCIInitialize()` with `OCI_EVENTS` mode to specify that the application is interested in registering for and receiving notifications. This starts a dedicated listening thread for notifications on the client.
2. Call `OCIHandleAlloc()` with handle type `OCI_HTYPE_SUBSCRIPTION` to allocate a subscription handle.
3. Call `OCIAttrSet()` to set the subscription handle attributes for:
   - `OCI_ATTR_SUBSCR_NAME` - subscription name
   - `OCI_ATTR_SUBSCR_NAMESPACE` - subscription namespace
   - `OCI_ATTR_SUBSCR_CALLBACK` - notification callback
   - `OCI_ATTR_SUBSCR_CTX` - callback context
   - `OCI_ATTR_SUBSCR_PAYLOAD` - payload buffer for posting
   - `OCI_ATTR_SUBSCR_RECPT` - recipient name
   - `OCI_ATTR_SUBSCR_RECPTPROTO` - protocol to receive notification with
   - `OCI_ATTR_SUBSCR_RECPTPRES` - presentation to receive notification with
   - `OCI_ATTR_SUBSCR_QOSFLAGS` - QOS (quality of service) levels
   - `OCI_ATTR_SUBSCR_TIMEOUT` - Registration timeout interval in seconds. The default is 0 if a timeout is not set.
   - `OCI_ATTR_SUBSCR_NTFN_GROUPING_CLASS` - notification grouping class

     Notifications can be spaced out by using the grouping NTFN option with the following constants. A value supported for notification grouping class is:

     ```
     #define OCI_SUBSCR_NTFN_GROUPING_CLASS_TIME   1 /* time  */
     ```

   - `OCI_ATTR_SUBSCR_NTFN_GROUPING_VALUE` - notification grouping value in seconds
   - `OCI_ATTR_SUBSCR_NTFN_GROUPING_TYPE` - notification grouping type

     Supported values for notification grouping type:

     ```
     #define OCI_SUBSCR_NTFN_GROUPING_TYPE_SUMMARY 1  /* summary */
     #define OCI_SUBSCR_NTFN_GROUPING_TYPE_LAST    2  /* last */
     ```

   - `OCI_ATTR_SUBSCR_NTFN_GROUPING_START_TIME` - notification grouping start time
   - `OCI_ATTR_SUBSCR_NTFN_GROUPING_REPEAT_COUNT` - notification grouping repeat count
   - If `OCI_SUBSCR_QOS_PURGE_ON_NTFN` is set, the registration is purged on the first notification.
   - If `OCI_SUBSCR_QOS_RELIABLE` is set, notifications are persistent. Surviving instances of a RAC can be used to send and retrieve change notification messages, even after a node failure because invalidations associated with this registration are queued persistently into the database. If `FALSE`, then invalidations are enqueued into a fast in-memory queue. Note that this option

describes the persistence of notifications and not the persistence of registrations. Registrations are automatically persistent by default.

`OCI_ATTR_SUBSCR_NAME`, `OCI_ATTR_SUBSCR_NAMESPACE` and `OCI_ATTR_SUBSCR_RECPTPROTO` must be set before registering a subscription.

If `OCI_ATTR_SUBSCR_RECPTPROTO` is set to `OCI_SUBSCR_PROTO_OCI`, then `OCI_ATTR_SUBSCR_CALLBACK` and `OCI_ATTR_SUBSCR_CTX` also need to be set.

If `OCI_ATTR_SUBSCR_RECPTPROTO` is set to `OCI_SUBSCR_PROTO_MAIL`, `OCI_SUBSCR_PROTO_SERVER`, or `OCI_SUBSCR_PROTO_HTTP`, then `OCI_ATTR_SUBSCR_RECPT` also needs to be set.

Setting `OCI_ATTR_SUBSCR_CALLBACK` and `OCI_ATTR_SUBSCR_RECPT` at the same time will cause an application error.

`OCI_ATTR_SUBSCR_PAYLOAD` is required before posting to a subscription.

See Also: "Subscription Handle Attributes" on page A-48 and "Creating the OCI Environment" on page 2-15 for setting up the environment with `mode = OCI_EVENTS | OCI_OBJECT`. `OCI_OBJECT` is required for grouping notifications.

4. The values of QOS, timeout interval, namespace, and port are set:

   See Also: "Setting QOS, Timeout Interval, Namespace, and Port Number" on page 9-53

5. If `OCI_ATTR_SUBSCR_RECPTPROTO` is set to `OCI_SUBSCR_PROTO_OCI`, then define the callback routine to be used with the subscription handle.

   See Also: "Notification Callback in OCI" on page 9-54

6. If `OCI_ATTR_SUBSCR_RECPTPROTO` is set to `OCI_SUBSCR_PROTO_SERVER`, then define the PL/SQL procedure, to be invoked on notification, in the database.

   See Also: "Notification Procedure" on page 9-56

7. Call `OCISubscriptionRegister()` to register with the subscriptions. This call can register interest in several subscriptions at the same time.

Example of Setting QOS Levels

```
/* Set QOS levels */
ub4 qosflags = OCI_SUBSCR_QOS_PAYLOAD;

/* Set QOS flags in subscription handle */
(void) OCIAttrSet((dvoid *) subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &qosflags, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_QOSFLAGS, errhp);

/* Set notification grouping class */
ub4 ntfn_grouping_class = OCI_SUBSCR_NTFN_GROUPING_CLASS_TIME;
(void) OCIAttrSet((dvoid *) subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &ntfn_grouping_class, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_NTFN_GROUPING_CLASS, errhp);

/* Set notification grouping value of 10 minutes */
ub4 ntfn_grouping_value = 600;
(void) OCIAttrSet((dvoid *) subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
```

```
                (dvoid *) &ntfn_grouping_value, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NTFN_GROUPING_VALUE, errhp);

/* Set notification grouping type */
ub4 ntfn_grouping_type = OCI_SUBSCR_NTFN_GROUPING_TYPE_SUMMARY;

/* Set notification grouping type in subscription handle */
(void) OCIAttrSet((dvoid *) subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) &ntfn_grouping_type, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NTFN_GROUPING_TYPE, errhp);

/* Set namespace specific context */
(void) OCIAttrSet((dvoid *) subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) NULL, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NAMESPACE_CTX, errhp);
```

Open Registration for Publish-Subscribe

Prerequisites for this method are:

- Registering using LDAP (open registration) requires the client to be an enterprise user.

  See Also: *Oracle Database Advanced Security Administrator's Guide*, sections on managing enterprise user security

- The compatibility of the database has to be 9.0 or higher.
- LDAP_REGISTRATION_ENABLED must be set to TRUE. This can be done this way:

  ```
  ALTER SYSTEM SET LDAP_REGISTRATION_ENABLED=TRUE
  ```

  The default is `FALSE`.

- `LDAP_REG_SYNC_INTERVAL` must be set to the time interval (in seconds) to refresh registrations from LDAP:

  ```
  ALTER SYSTEM SET LDAP_REG_SYNC_INTERVAL =  time_interval
  ```

  The default is 0, which means do not refresh.

- To force a database refresh of LDAP registration information immediately:

  ```
  ALTER SYSTEM REFRESH LDAP_REGISTRATION
  ```

The steps in open registration using Oracle Enterprise Security Manager (OESM) are:

1. In each enterprise domain, create enterprise role, ENTERPRISE_AQ_USER_ROLE.
2. For each database in the enterprise domain, add global role GLOBAL_AQ_USER_ROLE to enterprise role ENTERPRISE_AQ_USER_ROLE.
3. For each enterprise domain, add enterprise role ENTERPRISE_AQ_USER_ROLE to privilege group cn=OracleDBAQUsers, under cn=oraclecontext, under the administrative context.
4. For each enterprise user that is authorized to register for events in the database, grant enterprise role ENTERPRISE_AQ_USER_ROLE.

Using OCI to Open Register with LDAP

1. Call `OCIInitialize()` with `mode` set to `OCI_EVENTS | OCI_USE_LDAP`.

2. Call `OCIAttrSet()` to set the following environment handle attributes for accessing LDAP:
   - `OCI_ATTR_LDAP_HOST`: the host name on which the LDAP server resides
   - `OCI_ATTR_LDAP_PORT`: the port on which the LDAP server is listening
   - `OCI_ATTR_BIND_DN`: the distinguished name to login to the LDAP server, usually the DN of the enterprise user
   - `OCI_ATTR_LDAP_CRED`: the credential used to authenticate the client, for example, the password for simple authentication (user name and password)
   - `OCI_ATTR_WALL_LOC`: for SSL authentication, the location of the client wallet
   - `OCI_ATTR_LDAP_AUTH`: the authentication method code

     See Also: "Environment Handle Attributes" on page A-2 for a complete list of authentication modes

   - `OCI_ATTR_LDAP_CTX`: the administrative context for Oracle in the LDAP server
3. Call `OCIHandleAlloc()` with handle type `OCI_HTYPE_SUBSCRIPTION`, to allocate a subscription handle.
4. Call `OCIDescriptorAlloc()` with descriptor type `OCI_DTYPE_SRVDN`, to allocate a server DN descriptor.
5. Call `OCIAttrSet()` to set the server DN descriptor attributes for `OCI_ATTR_SERVER_DN`, the distinguished name of the database in which the client wants to receive notifications. `OCIAttrSet()` can be called multiple times for this attribute so that more than one database server is included in the registration
6. Call `OCIAttrSet()` to set the subscription handle attributes for:
   - `OCI_ATTR_SUBSCR_NAME` - subscription name
   - `OCI_ATTR_SUBSCR_NAMESPACE` - subscription namespace
   - `OCI_ATTR_SUBSCR_CALLBACK` - notification callback
   - `OCI_ATTR_SUBSCR_CTX` - callback context
   - `OCI_ATTR_SUBSCR_PAYLOAD` - payload buffer for posting
   - `OCI_ATTR_SUBSCR_RECPT` - recipient name
   - `OCI_ATTR_SUBSCR_RECPTPROTO` - protocol to receive notification
   - `OCI_ATTR_SUBSCR_RECPTRES` - presentation to receive notification with
   - `OCI_ATTR_SUBSCR_QOSFLAGS` - QOS (quality of service) levels
   - `OCI_ATTR_SUBSCR_TIMEOUT` - Registration timeout interval in seconds. The default is 0 if a timeout is not set.
   - `OCI_ATTR_SUBSCR_SERVER_DN` - the descriptor handles you populated in step 5
7. The values of QOS, timeout interval, namespace, and port are set:

   See Also: "Setting QOS, Timeout Interval, Namespace, and Port Number" on page 9-53

8. Call `OCISubscriptionRegister()` to register the subscriptions. The registration will take effect when the database accesses LDAP to pick up new registrations. The frequency of pick-ups is determined by the value of REG_SYNC_INTERVAL.

Setting QOS, Timeout Interval, Namespace, and Port Number

You can set QOSFLAGS to the following QOS levels using `OCIAttrSet()`:

- `OCI_SUBSCR_QOS_RELIABLE` - Reliable notification persists across instance and database restarts. Reliability is of the server only and is only for persistent queues or buffered messages. This option describes the persistence of the notifications. Registrations are persistent by default.
- `OCI_SUBSCR_QOS_PURGE_ON_NTFN` - Once received, purge registration on first notification. (Subscription is unregistered.)

```
/* Set QOS levels */
ub4 qosflags = OCI_SUBSCR_QOS_RELIABLE | OCI_SUBSCR_QOS_PURGE_ON_NTFN;

/* Set flags in subscription handle */
(void)OCIAttrSet((void *)subscrhp, (ub4)OCI_HTYPE_SUBSCRIPTION,
                 (void *)&qosflags, (ub4)0, (ub4)OCI_ATTR_SUBSCR_QOSFLAGS, errhp);

/* Set auto-expiration after 30 seconds */
ub4 timeout = 30;
(void)OCIAttrSet((void *)subscrhp, (ub4)OCI_HTYPE_SUBSCRIPTION,
                  (void *)&timeout, (ub4)0, (ub4)OCI_ATTR_SUBSCR_TIMEOUT, errhp);
```

The registration is purged when the timeout is exceeded, and a notification is sent to the client, so that the client can invoke its callback and take any necessary action. In the case of client failure before the timeout, the registration is purged.

You can set the port number on the environment handle, which is important if the client is on a system behind a firewall that is able to receive notifications only on certain ports. Clients can specify the port for the listener thread before the first registration, using an attribute in the environment handle. The thread is started the first time `OCISubscriptionRegister()` is called. If available, this specified port number will be used. An error is returned if the client tries to start another thread on a different port using a different environment handle.

```
ub4 port = 1581;
(void)OCIAttrSet((void *)envhp, (ub4)OCI_HTYPE_ENV, (void *)&port, (ub4)0,
                  (ub4)OCI_ATTR_SUBSCR_PORTNO, errhp);
```

If instead, the port is determined automatically, you can get the port number at which the client thread is listening for notification by obtaining the attribute from the environment handle.

```
(void)OCIAttrGet((void *)subhp, (ub4)OCI_HTYPE_ENV, (void *)&port, (ub4)0,
                  (ub4)OCI_ATTR_SUBSCR_PORTNO, errhp);
```

OCI Functions Used to Manage Publish-Subscribe Notification

The following functions are used to manage publish-subscribe notification.

Table 9–13 Publish-Subscribe Functions

Function	Purpose
OCISubscriptionDisable()	Disables a subscription.

Table 9–13 (Cont.) Publish-Subscribe Functions

Function	Purpose
OCISubscriptionEnable()	Enables a subscription.
OCISubscriptionPost()	Posts a subscription.
OCISubscriptionRegister()	Registers a subscription.
OCISubscriptionUnRegister()	Unregisters a subscription.

Notification Callback in OCI

The client needs to register a notification callback that gets invoked when there is some activity on the subscription for which interest has been registered. In the AQ namespace, for instance, this occurs when a message of interest is enqueued.

This callback is typically set through the `OCI_ATTR_SUBSCR_CALLBACK` attribute of the subscription handle.

See Also: "Subscription Handle Attributes" on page A-48

The callback must return a value of `OCI_CONTINUE` and adhere to the following specification:

```
typedef ub4 (*OCISubscriptionNotify) ( void            *pCtx,
                                       OCISubscription *pSubscrHp,
                                       void            *pPayload,
                                       ub4             iPayloadLen,
                                       void            *pDescriptor,
                                       ub4             iMode);
```

The parameters are described as follows:

pCtx (IN)
A user-defined context specified when the callback was registered.

pSubscrHp (IN)
The subscription handle specified when the callback was registered.

pPayload (IN)
The payload for this notification. For this release, only ub1 * (a sequence of bytes) for the payload is supported.

iPayloadLen (IN)
The length of the payload for this notification.

pDescriptor (IN)
The namespace-specific descriptor. Namespace-specific parameters can be extracted from this descriptor. The structure of this descriptor is opaque to the user and its type is dependent on the namespace.

The attributes of the descriptor are namespace-specific. For Advanced Queuing (AQ), the descriptor is `OCI_DTYPE_AQNFY`. For the AQ namespace, the count of notifications received in the group is provided in the notification descriptor. The attributes of `pDescriptor` are:

- Notification Flag (regular = 0, timeout = 1, or grouping notification = 2) - `OCI_ATTR_NFY_FLAGS`
- Queue Name - `OCI_ATTR_QUEUE_NAME`

- Consumer Name - OCI_ATTR_CONSUMER_NAME
- Message Id - OCI_ATTR_NFY_MSGID
- Message Properties - OCI_ATTR_MSG_PROP
- Count of Notifications Received In the Group - OCI_ATTR_AQ_NTFN_GROUPING_COUNT
- The Group, an OCI Collection - OCI_ATTR_AQ_NTFN_GROUPING_MSGID_ARRAY

See Also:

- "OCI and Streams Advanced Queuing" on page 9-41
- "Notification Descriptor Attributes" on page A-56

iMode (IN)

Call-specific mode. Valid value:

- OCI_DEFAULT - executes the default call

Example of OCI Notification Callback for AQ Grouping

Here is an example of how to use the new AQ grouping notification attributes in a notification callback.

```
ub4 notifyCB1(void *ctx, OCISubscription *subscrhp, void *pay, ub4 payl,
              void *desc, ub4 mode)
{
 oratext            *subname;
 ub4                 size;
 OCIColl            *msgid_array = (OCIColl *)0;
 ub4                 msgid_cnt = 0;
 OCIRaw             *msgid;
 void               **msgid_ptr;
 sb4                 num_msgid = 0;
 void               *elemind = (void *)0;
 boolean             exist;
 ub2                 flags;
 oratext            *hexit = (oratext *)"0123456789ABCDEF";
 ub4                 i, j;

 /* get subscription name */
 OCIAttrGet(subscrhp, OCI_HTYPE_SUBSCRIPTION, (void *)&subname, &size,
            OCI_ATTR_SUBSCR_NAME,ctxptr->errhp);

 /* print subscripton name */
 printf("Got notification for %.*s\n", size, subname);
 fflush((FILE *)stdout);

 /* get the #ntfns received in this group */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY, (void *)&msgid_cnt, &size,
            OCI_ATTR_AQ_NTFN_GROUPING_COUNT, ctxptr->errhp);

 /* get the group - collection of msgids */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY, (void *)&msgid_array, &size,
            OCI_ATTR_AQ_NTFN_GROUPING_MSGID_ARRAY, ctxptr->errhp);

 /* get notification flag - regular, timeout, or grouping notification? */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY, (void *)&flags, &size,
            OCI_ATTR_NFY_FLAGS, ctxptr->errhp);
```

```
/* print notification flag */
printf("Flag: %d\n", (int)flags);

/* get group (collection) size */
if (msgid_array)
  checkerr(ctxptr->errhp,
       OCICollSize(ctxptr->envhp, ctxptr->errhp,
       CONST OCIColl *) msgid_array, &num_msgid),
       "Inside notifyCB1-OCICollSize");
else
  num_msgid =0;

/* print group size */
printf("Collection size: %d\n", num_msgid);

/* print all msgids in the group */
for(i = 0; i < num_msgid; i++)
{
  ub4  rawSize;                                          /* raw size    */
  ub1 *rawPtr;                                           /* raw pointer */
    /* get msgid from group */
  checkerr(ctxptr->errhp,
       OCICollGetElem(ctxptr->envhp, ctxptr->errhp,
              (OCIColl *) msgid_array, i, &exist,
              (void **)(&msgid_ptr), &elemind),
       "Inside notifyCB1-OCICollGetElem");
  msgid = *msgid_ptr;
  rawSize = OCIRawSize(ctxptr->envhp, msgid);
  rawPtr = OCIRawPtr(ctxptr->envhp, msgid);

  /* print msgid size */
  printf("Msgid size: %d\n", rawSize);

  /* print msgid in hex format */
  for (j = 0; j < rawSize; j++)
  {                                          /* for each byte in the raw */
    printf("%c", hexit[(rawPtr[j] & 0xf0) >> 4]);
    printf("%c", hexit[(rawPtr[j] & 0x0f)]);
  }
  printf("\n");
}

/* print #ntfns received in group */
printf("Notification Count: %d\n", msgid_cnt);
printf("\n");
printf("************************************************************\n");
fflush((FILE *)stdout);
return 1;
}
```

Notification Procedure

The PL/SQL procedure that will be invoked when there is some activity on the subscription for which interest has been registered, has to be created in the database.

This procedure is typically set through the `OCI_ATTR_SUBSCR_RECPT` attribute of the subscription handle.

See Also:

- See "Subscription Handle Attributes" on page A-48.
- "Oracle Streams AQ PL/SQL Callback" in *Oracle Database PL/SQL Packages and Types Reference* for the PL/SQL procedure specification.

Publish-Subscribe Direct Registration Example

This example shows how system events, client notification, and Advanced Queuing work together to implement publish subscription notification.

The following PL/SQL code creates all objects necessary to support a publish-subscribe mechanism under the user schema `pubsub`. In this code, the Agent `snoop` subscribes to messages that are published at logon events. Note that the user `pubsub` needs `AQ_ADMINISTRATOR_ROLE` and `AQ_USER_ROLE` privileges to use Advance Queuing functionality. The initialization parameter `_SYSTEM_TRIG_ENABLED` must be set to `TRUE` (the default) to enable triggers for system events. Connect as `pubsub` before running the following code:

```
-------------------------------------------------------------
----create queue table for persistent multiple consumers
-------------------------------------------------------------
---- Create or replace a queue table
begin
  DBMS_AQADM.CREATE_QUEUE_TABLE(
  QUEUE_TABLE=>'pubsub.raw_msg_table',
  MULTIPLE_CONSUMERS => TRUE,
  QUEUE_PAYLOAD_TYPE =>'RAW',
  COMPATIBLE => '8.1.5');
end;
/
-------------------------------------------------------------
---- Create a persistent queue for publishing messages
-------------------------------------------------------------
---- Create a queue for logon events
begin
  DBMS_AQADM.CREATE_QUEUE(QUEUE_NAME=>'pubsub.logon',
  QUEUE_TABLE=>'pubsub.raw_msg_table',
  COMMENT=>'Q for error triggers');
end;
/
-------------------------------------------------------------
---- Start the queue
-------------------------------------------------------------
begin
  DBMS_AQADM.START_QUEUE('pubsub.logon');
end;
/
-------------------------------------------------------------
---- define new_enqueue for convenience
-------------------------------------------------------------
create or replace procedure new_enqueue(queue_name  in varchar2,
                                        payload  in raw ,
correlation in varchar2 := NULL,
exception_queue in varchar2 := NULL)
as
  enq_ct     dbms_aq.enqueue_options_t;
  msg_prop   dbms_aq.message_properties_t;
  enq_msgid  raw(16);
```

```
  userdata   raw(1000);
begin
  msg_prop.exception_queue := exception_queue;
  msg_prop.correlation := correlation;
  userdata := payload;
  DBMS_AQ.ENQUEUE(queue_name,enq_ct, msg_prop,userdata,enq_msgid);
end;
/
-------------------------------------------------------------
---- add subscriber with rule based on current user name,
---- using correlation_id
-------------------------------------------------------------
declare
subscriber sys.aq$_agent;
begin
  subscriber := sys.aq$_agent('SNOOP', null, null);
  dbms_aqadm.add_subscriber(queue_name => 'pubsub.logon',
                            subscriber => subscriber,
                            rule => 'CORRID = ''ix'' ');
end;
/
-------------------------------------------------------------
---- create a trigger on logon on database
-------------------------------------------------------------
---- create trigger on after logon
create or replace trigger systrig2
   AFTER LOGON
   ON DATABASE
   begin
     new_enqueue('pubsub.logon', hextoraw('9999'), dbms_standard.login_user);
   end;
/

-------------------------------------------------------------
---- create a PL/SQL callback for notification of logon
---- of user 'ix' on database
-------------------------------------------------------------
----
create or replace procedure plsqlnotifySnoop(
  context raw, reginfo sys.aq$_reg_info, descr sys.aq$_descriptor,
  payload raw, payloadl number)
as
begin
 dbms_output.put_line('Notification : User ix Logged on\n');
end;
/
```

After the subscriptions are created, the client needs to register for notification using callback functions. The following sample code performs the necessary steps for registration. The initial steps of allocating and initializing session handles are omitted here for sake of clarity.

```
...
static ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;

static OCISubscription *subscrhpSnoop = (OCISubscription *)0;
static OCISubscription *subscrhpSnoopMail = (OCISubscription *)0;
static OCISubscription *subscrhpSnoopServer = (OCISubscription *)0;

/* callback function for notification of logon of user 'ix' on database */
```

```
static ub4 notifySnoop(ctx, subscrhp, pay, payl, desc, mode)
    void *ctx;
    OCISubscription *subscrhp;
    void *pay;
    ub4 payl;
    void *desc;
    ub4 mode;
{
    printf("Notification : User ix Logged on\n");
  (void)OCIHandleFree((void *)subscrhpSnoop,
            (ub4) OCI_HTYPE_SUBSCRIPTION);
    return 1;
}

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  ub4 buflen;
  sb4 errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_SUCCESS_WITH_INFO:
    printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    printf("Error - OCI_NO_DATA\n");
    break;
  case OCI_ERROR:
    OCIErrorGet ((void *) errhp, (ub4) 1, (text *) NULL, &errcode,
            errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  case OCI_STILL_EXECUTING:
    printf("Error - OCI_STILL_EXECUTING\n");
    break;
  case OCI_CONTINUE:
    printf("Error - OCI_CONTINUE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
}

static void initSubscriptionHn (subscrhp,
                          subscriptionName,
                          func,
                          recpproto,
```

```
                        recpaddr,
                        recppres)
OCISubscription **subscrhp;
  char * subscriptionName;
  void * func;
  ub4 recpproto;
  char * recpaddr;
  ub4 recppres;
{
    /* allocate subscription handle */
    (void) OCIHandleAlloc((void *) envhp, (void **)subscrhp,
        (ub4) OCI_HTYPE_SUBSCRIPTION,
        (size_t) 0, (void **) 0);

    /* set subscription name in handle */
    (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (void *) subscriptionName,
        (ub4) strlen((char *)subscriptionName),
        (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

    /* set callback function in handle */
    if (func)
      (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
          (void *) func, (ub4) 0,
          (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

    /* set context in handle */
    (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (void *) 0, (ub4) 0,
       (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

    /* set namespace in handle */
    (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (void *) &namespace, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

    /* set receive with protocol in handle */
    (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (void *) &recpproto, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_RECPTPROTO, errhp);

    /* set recipient address in handle */
    if (recpaddr)
      (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
          (void *) recpaddr, (ub4) strlen(recpaddr),
          (ub4) OCI_ATTR_SUBSCR_RECPT, errhp);

    /* set receive with presentation in handle */
    (void) OCIAttrSet((void *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (void *) &recppres, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_RECPTPRES, errhp);

    printf("Begining Registration for subscription %s\n", subscriptionName);
    checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 1, errhp,
        OCI_DEFAULT));
   printf("done\n");

}
```

```
int main( argc, argv)
int    argc;
char * argv[];
{
    OCISession *authp = (OCISession *) 0;

/*************************************************
Initialize OCI Process/Environment
Initialize Server Contexts
Connect to Server
Set Service Context
*************************************************/

/* Registration Code Begins */
/* Each call to initSubscriptionHn allocates
   and Initialises a Registration Handle */

/* Register for OCI notification */
    initSubscriptionHn(    &subscrhpSnoop,    /* subscription handle*/
    (char*) "PUBSUB.LOGON:SNOOP", /* subscription name */
                                  /*<queue_name>:<agent_name> */
        (void*)notifySnoop,  /* callback function */
        OCI_SUBSCR_PROTO_OCI, /* receive with protocol */
        (char *)0, /* recipient address */
        OCI_SUBSCR_PRES_DEFAULT); /* receive with presentation */

/* Register for email notification */
    initSubscriptionHn(    &subscrhpSnoopMail,  /* subscription handle */
     (char*) "PUBSUB.LOGON:SNOOP",              /* subscription name */
                                                /* <queue_name>:<agent_name> */
        (void*)0, /* callback function */
        OCI_SUBSCR_PROTO_MAIL, /* receive with protocol */
        (char*)  "longying.zhao@oracle.com", /* recipient address */
        OCI_SUBSCR_PRES_DEFAULT); /* receive with presentation */

/* Register for server to server notification */
    initSubscriptionHn(    &subscrhpSnoopServer, /* subscription handle */
       (char*)  "PUBSUB.LOGON:SNOOP",            /* subscription name */
                                                 /* <queue_name>:<agent_name> */
        (void*)0, /* callback function */
        OCI_SUBSCR_PROTO_SERVER, /* receive with protocol */
         (char*) "pubsub.plsqlnotifySnoop", /* recipient address */
        OCI_SUBSCR_PRES_DEFAULT); /* receive with presentation */

    checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) OCI_DEFAULT));

/*************************************************
The Client Process does not need a live Session for Callbacks
End Session and Detach from Server
*************************************************/

    OCISessionEnd ( svchp,  errhp, authp, (ub4) OCI_DEFAULT);

    /* detach from server */
    OCIServerDetach( srvhp, errhp, OCI_DEFAULT);

   while (1)    /* wait for callback */
    sleep(1);
}
```

If user IX logs on to the database, the client is notified by e-mail, and the callback function notifySnoop is called. An e-mail notification will be sent to the address xyz@company.com and the PL/SQL procedure plsqlnotifySnoop will also be called in the database.

Publish-Subscribe LDAP Registration Example

The following code fragment illustrates how to do LDAP registration. Please read all the program comments:

```
...

  /* TO use LDAP registration feature, OCI_EVENTS | OCI_EVENTS |OCI_USE_LDAP must
  /*   be set in OCIInitialize: */
  (void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT|OCI_USE_LDAP, (void *)0,
                       (void * (*)(void *, size_t)) 0,
                       (void * (*)(void *, void *, size_t))0,
                       (void (*)(void *, void *)) 0 );

...

  /* set LDAP attributes in the environment handle */

  /* LDAP host name */
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)"yow", 3,
                    OCI_ATTR_LDAP_HOST, (OCIError *)errhp);

  /* LDAP server port */
  ldap_port = 389;
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)&ldap_port,
                    (ub4)0, OCI_ATTR_LDAP_PORT, (OCIError *)errhp);

  /* bind DN of the client, normally the enterprise user name */
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)"cn=orcladmin",
                    12, OCI_ATTR_BIND_DN, (OCIError *)errhp);

  /* password of the client */
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)"welcome",
                    7, OCI_ATTR_LDAP_CRED, (OCIError *)errhp);

  /* authentication method is "simple", username/password authentication */
  ldap_auth = 0x01;
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)&ldap_auth,
                    (ub4)0, OCI_ATTR_LDAP_AUTH, (OCIError *)errhp);

  /* adminstrative context: this is the DN above cn=oraclecontext */
  (void) OCIAttrSet((void *)envhp, OCI_HTYPE_ENV, (void *)"cn=acme,cn=com",
                    14, OCI_ATTR_LDAP_CTX, (OCIError *)errhp);

...

  /* retrieve the LDAP attributes from the environment handle */

  /* LDAP host */
  (void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&buf,
                    &szp,  OCI_ATTR_LDAP_HOST,  (OCIError *)errhp);

  /* LDAP server port */
  (void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&intval,
```

```
                0,  OCI_ATTR_LDAP_PORT,  (OCIError *)errhp);

/* client binding DN */
(void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&buf,
                &szp,  OCI_ATTR_BIND_DN,  (OCIError *)errhp);

/* client password */
(void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&buf,
                &szp,  OCI_ATTR_LDAP_CRED,  (OCIError *)errhp);

/* adminstrative context */
(void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&buf,
                &szp,  OCI_ATTR_LDAP_CTX,  (OCIError *)errhp);

/* client authentication method */
(void) OCIAttrGet((void *)envhp, OCI_HTYPE_ENV, (void *)&intval,
                0,  OCI_ATTR_LDAP_AUTH,  (OCIError *)errhp);

...

/* to set up the server DN descriptor in the subscription handle */

/* allocate a server DN descriptor, dn is of type "OCIServerDNs **",
   subhp is of type "OCISubscription **" */
(void) OCIDescriptorAlloc((void *)envhp, (void **)dn,
                    (ub4) OCI_DTYPE_SRVDN, (size_t)0, (void **)0);

/* now *dn is the server DN descriptor, add the DN of the first database
   that we want to register */
(void) OCIAttrSet((void *)*dn, (ub4) OCI_DTYPE_SRVDN,
                (void *)"cn=server1,cn=oraclecontext,cn=acme,cn=com",
                42, (ub4)OCI_ATTR_SERVER_DN, errhp);
/* add the DN of another database in the descriptor */
(void) OCIAttrSet((void *)*dn, (ub4) OCI_DTYPE_SRVDN,
                (void *)"cn=server2,cn=oraclecontext,cn=acme,cn=com",
                42, (ub4)OCI_ATTR_SERVER_DN, errhp);

/* set the server DN descriptor into the subscription handle */
(void) OCIAttrSet((void *) *subhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
              (void *) *dn, (ub4)0, (ub4) OCI_ATTR_SERVER_DNS, errhp);

...

/* now we will try to get the server DN information from the subscription
   handle */

/* first, get the server DN descriptor out */
(void) OCIAttrGet((void *) *subhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
                (void *)dn, &szp, OCI_ATTR_SERVER_DNS, errhp);

/* then, get the number of server DNs in the descriptor */
(void) OCIAttrGet((void *) *dn, (ub4)OCI_DTYPE_SRVDN, (void *)&intval,
                &szp, (ub4)OCI_ATTR_DN_COUNT, errhp);

/* allocate an array of char * to hold server DN pointers returned by
   oracle */
  if (intval)
  {
    arr = (char **)malloc(intval*sizeof(char *));
    (void) OCIAttrGet((void *)*dn, (ub4)OCI_DTYPE_SRVDN, (void *)arr,
```

```
                        &intval, (ub4)OCI_ATTR_SERVER_DN, errhp);
    }

  /* OCISubscriptionRegister() calls have two modes: OCI_DEFAULT and
     OCI_REG_LDAPONLY. If OCI_DEFAULT is used, there should be only one
     server DN in the server DN descriptor. The registration request will
     be sent to the database. If a database connection is not available,
     the registration request will be detoured to the LDAP server. On the
     other hand, if mode OCI_REG_LDAPONLY is used the registration request
     will be directly sent to LDAP. This mode should be used when there are
     more than one server DNs in the server DN descriptor, or we are sure
     that a database connection is not available.

     In this example, two DNs are entered; so we should use mode
     OCI_REG_LDAPONLY in register. */
  OCISubscriptionRegister(svchp, subhp, 1, errhp, OCI_REG_LDAPONLY);

  ...

  /* as OCISubscriptionRegister(), OCISubscriptionUnregister() also has
     mode OCI_DEFAULT and OCI_REG_LDAPONLY. The usage is the same. */

  OCISubscriptionUnRegister(svchp, *subhp, errhp, OCI_REG_LDAPONLY);
}
...
```

Exhibit 17

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY