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 IN SUPPORT OF EPICREALM'S MOTION FOR PARTIAL SUMMARY JUDGMENT OF LITERAL INFRINGEMENT

VOLUME III of III

EXHIBITS 21 to 32

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: July 31, 2008
Public Version Dated: August 7, 2008
876825 / 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.	Excerpt from Report of Plaintiffs' Expert Michael I. Shamos Concerning Invalidity, May 2, 2008
4.	Excerpt from Oracle WebServer User's Guide, Release 1.0 A34986-2, ORCL00941713-00941960
5.	The Oracle WebServer: A Technical Discussion, ORCL01042968-01042974
6.	Oracle WebServer Release 2.0, Feature List, 10/30/95, ORCL00937853-00937862
7.	Excerpt from Expert Witness Rebuttal Report on Non-Infringement By Dr. Paul C. Clark, June 6, 2008
8.	Excerpt from Oracle Database Application Guide – Fundamentals, 10g Release 2 (10.2), B14251-01
9.	Excerpt from Oracle Application Server Containers for J2EE User's Guide 10g (9.0.4) B10322-01, ORCL00338416-00338661
10.	Parties' Proposed Element-by-Element Claim Construction Chart, June 30, 2008
11.	Excerpt from Oracle Application Server Web Cache Administrator's Guide, 10g Release 2 (10.1.2), B14046-05, ORCL00513887-00514436
12.	Excerpt from Oracle XML DB Developer's Guide 10g Release 2 (10.2), B14259-02.
13.	Excerpt from Oracle Real Application Clusters Administration and Deployment Guide, 11g Release 1 (11.1), B28254-04, ORCL01972334-01972630
14.	What's New for Java DB, JDBC, and Database Web Servers in Oracle Database 10g, an Oracle White Paper, May 2005
15.	Excerpt from Oracle Database New Features Guide, 11g Release 1 (11.1) B28279-02
16.	Functional Specification for Runtime Load Balancing, RDBMS, 10gR2, ORCL01060148-01060193
17.	Excerpt from Plaintiff's Responses to Defendants' First Set of Interrogatories (1-6)
18.	Oracle World, ORCL01050220-01050280

Exh.	Document
19.	Excerpt from Oracle Application Server 10g Concepts, 10g (9.0.4), B10375-02, ORCL00370687-00370914
20.	Oracle9i Application Server, An Oracle White Paper, September 2002, ORCL00010730-00010792
21.	Excerpt from Oracle Call Interface Programmer's Guide, 11g Release 1 (11.1), B28395-03
22.	Design Specification for OCI Runtime Connection Load Balancing in 11g, 12/14/08, ORCL01310468-01310498
23.	Functional/Design Specification for Runtime Work Request Load Balancing using RAC Service Metrics, JDBC, 10gR2, 5/1/04, ORCL01054844-01054858
24.	Oracle HTTP Server, mod_plsql User's Guide, 10g Release 2 (10.2), B14337-01
25.	Excerpt from deposition transcript of Paul C, Clark, July 11, 2008
26.	Excerpt from deposition transcript of Steve Harris, April 18, 2008
27.	Excerpt from deposition transcript of Pushkar Kapasi, February 27, 2008
28.	Excerpt from deposition transcript of Matthew Mayerson, April 15, 2008
29.	Excerpt from deposition transcript of Katrina Montinola, February 12, 2008
30.	Excerpt from deposition transcript of Mark Nelson, February 22, 2008
31.	Excerpt from deposition transcript of Douglas Surber, February 28, 2008
32.	Oracle10iAS Planning Update, ORCL01066383-01066413

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 7, 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 7, 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 21

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.

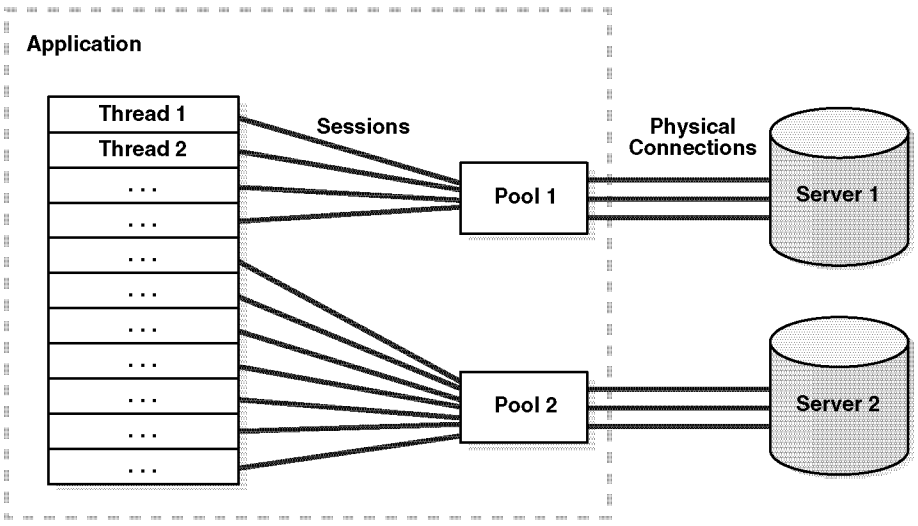
What connection pooling offers is stateless connections and stateful sessions. Users who need to work with stateless sessions, see "Session Pooling in OCI" on page 9-7.

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.
```

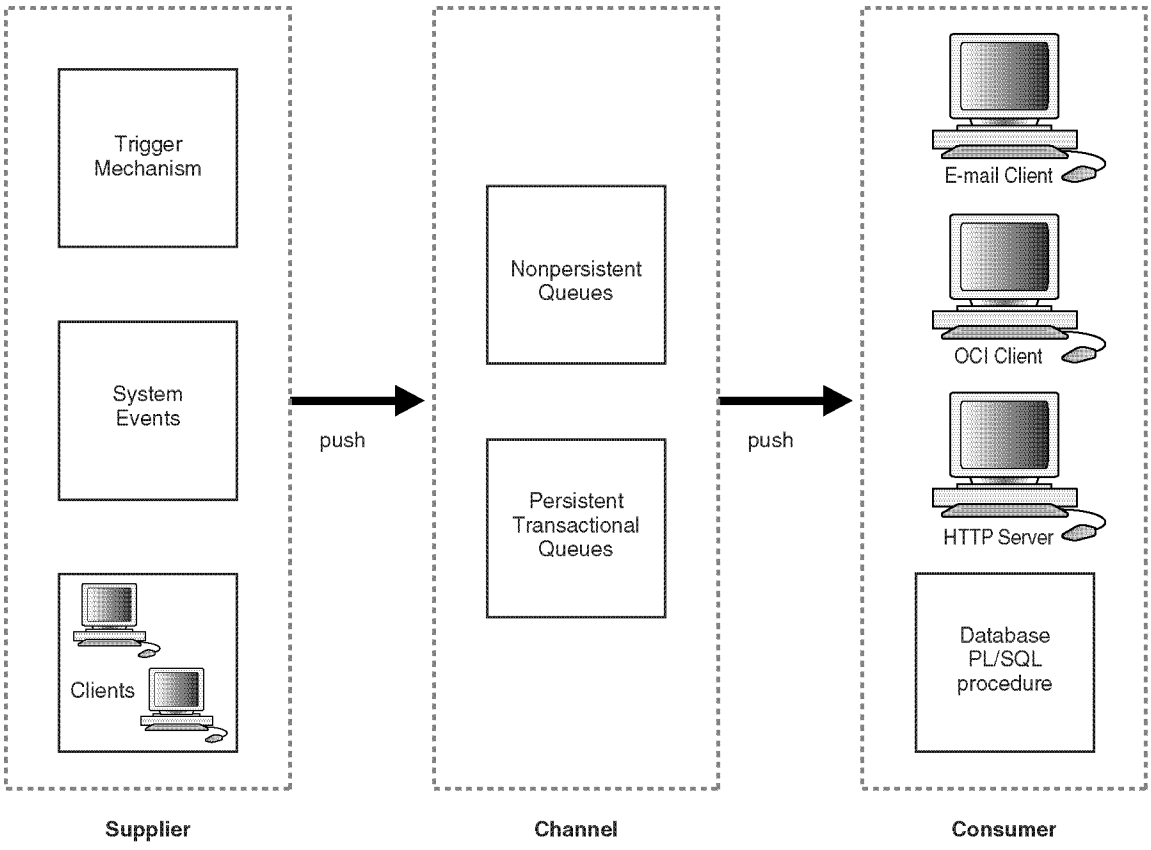
```
*/
consumer = "FIRST_SUBSCRIBER";
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (void *)consumer,
           (ub4)strlen((char*)consumer), OCI_ATTR_CONSUMER_NAME, errhp);
/* Dequeue the message but don't return the payload (to simplify the code
   snippet)
*/
OCIAQDeq(svchp, errhp,  "test_queue", deqopt, 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 22

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 23

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 24

Oracle® HTTP Server

mod_plsql User's Guide

10*g* Release 2 (10.2)

B14337-01

June 2005

Oracle HTTP Server mod_plsql User's Guide, 10*g* Release 2 (10.2)

B14337-01

Copyright © 1996, 2005, Oracle. All rights reserved.

Primary Author: Pravin Prabhakar

Contributing Author: Peter Lubbers

Contributors: Pushkar Kapasi, Eric Lee, and Thomas Van Raalte

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.

Contents

Preface ix

Audience ix
Documentation Accessibility ix
Related Documentation x
Conventions x

1 Understanding mod_plsql

1.1 Processing Client Requests 1-1
1.2 Database Access Descriptors (DADs) 1-2
1.3 Invoking mod_plsql 1-3
1.4 Transaction Mode 1-5
1.5 Supported Data Types 1-5
1.6 Parameter Passing 1-5
1.6.1 Parameter Passing by Name (Overloaded Parameters) 1-6
1.6.2 Flexible Parameter Passing 1-7
1.6.2.1 Two Parameter Interface 1-7
1.6.2.2 Four Parameter Interface 1-8
1.6.3 Large Parameter Passing 1-9
1.7 File Upload and Download 1-9
1.7.1 Document Table Definition 1-10
1.7.1.1 Semantics of the CONTENT Column 1-11
1.7.1.2 Semantics of the CONTENT_TYPE Column 1-11
1.7.1.3 Semantics of the LAST_UPDATED Column 1-11
1.7.1.4 Semantics of the DAD_CHARSET Column 1-11
1.7.2 Old Style Document Table Definition 1-11
1.7.3 Configuration Parameters for Document Upload/Downloading 1-11
1.7.3.1 PlsqlDocumentTablename 1-12
1.7.3.2 PlsqlDocumentPath (Document Access Path) 1-12
1.7.3.3 PlsqlDocumentProcedure (Document Access Procedure) 1-13
1.7.3.4 PlsqlUploadAsLongRaw 1-13
1.7.4 File Upload 1-13
1.7.5 Specifying Attributes (Mime Types) of Uploaded Files 1-15
1.7.6 Uploading Multiple Files 1-15
1.7.7 File Download 1-15

1.7.8 Direct BLOB Download ... 1-16
1.8 Path Aliasing (Direct Access URLs) ... 1-17
1.9 Common Gateway Interface (CGI) Environment Variables ... 1-17
1.9.1 Adding and Overriding CGI Environment Variables ... 1-18
1.9.2 PlsqlNLSLanguage ... 1-19
1.9.2.1 REQUEST_CHARSET CGI Environment Variable ... 1-19
1.9.2.2 REQUEST_IANA_CHARSET CGI Environment Variable ... 1-20
1.10 Restrictions in mod_plsql ... 1-20

2 Configuring mod_plsql

2.1 Verifying Requirements ... 2-1
2.2 Installing Required Packages ... 2-1
2.3 Creating a DAD for Your PL/SQL Application ... 2-3

3 Securing Application Database Access Through mod_plsql

3.1 Authenticating Users Through mod_plsql ... 3-1
3.1.1 Basic (Database Controlled Authentication) ... 3-2
3.1.2 Oracle HTTP Server mod_plsql Basic Authentication Mode ... 3-2
3.1.3 Global OWA, Custom OWA, and Per Package (Custom Authentication) ... 3-2
3.2 Deauthenticating Users ... 3-3
3.3 Protecting the PL/SQL Procedures Granted to PUBLIC ... 3-3
3.3.1 Using the PlsqlExclusionList Directive in mod_plsql ... 3-4
3.3.2 Accessing the PlsqlExclusionList Directive ... 3-4

4 Optimizing PL/SQL Performance

4.1 PL/SQL Performance in Oracle HTTP Server - Overview ... 4-1
4.2 Process-Based and Thread-Based Operation in Oracle HTTP Server ... 4-2
4.3 Performance Tuning Issues for mod_plsql ... 4-3
4.3.1 Connection Pooling with mod_plsql ... 4-3
4.3.2 Closing Pooled Database Sessions ... 4-4
4.3.3 Restarting the mod_plsql Connection Pool ... 4-5
4.4 Performance Tuning Areas in mod_plsql ... 4-6
4.4.1 PL/SQL Based Web Application Development Considerations ... 4-6
4.4.2 Connection Pooling Tips and Oracle HTTP Server Configuration ... 4-7
4.4.3 Tuning the Number of Database Sessions ... 4-9
4.4.4 Two-Listener Strategy ... 4-9
4.4.5 Overhead Problems ... 4-10
4.4.5.1 The Describe Overhead ... 4-11
4.4.5.2 Avoiding the Describe Overhead ... 4-11
4.4.6 The Flexible Parameter Passing (four-parameter) Overhead ... 4-12
4.5 Using Caching with PL/SQL-Based Web Applications ... 4-12
4.5.1 Using the Validation Technique ... 4-13
4.5.1.1 Last-Modified ... 4-13
4.5.1.2 Entity Tag Method ... 4-13
4.5.1.3 Using the Validation Technique for mod_plsql ... 4-13
4.5.1.4 Second Request Using the Validation Technique ... 4-14

4.5.2 Using the Expires Technique 4-15
4.5.3 System- and User-Level Caching with PL/SQL-Based Web Applications 4-17
4.6 Tuning File System Cache to Improve Caching Performance 4-18
4.6.1 Introducing File System Cache Tuning 4-19
4.6.2 Enabling File System Cache 4-20
4.6.3 Configuring File System Cache to Reside on a Faster File System 4-20
4.6.4 Resizing File System Cache 4-20
4.6.4.1 Setting the Total Cache Size with PlsqlCacheTotalSize 4-21
4.6.4.2 Setting the Days of Aging for Cache with PlsqlCacheMaxAge 4-22
4.6.4.3 Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize 4-22
4.6.5 Configuring Cache Cleanup 4-22
4.7 Oracle HTTP Server Directives 4-23

A Frequently Asked Questions

Index

List of Examples

1–1 Invoking A Procedure That Does Not Take Arguments 1-3
1–2 Invoking A Procedure That Takes Arguments 1-4
1–3 Invoking the Default Procedure Stored in the DAD Configuration 1-4
1–4 Sending a URL to Execute the Scalar Version of a Procedure 1-7
1–5 Sending a URL to Execute the Array Version of a Procedure 1-7
1–6 Two Parameter Interface 1-8
1–7 Four Parameter Interface 1-9
1–8 Parameters for Document Upload/Download 1-12
1–9 PlsqlDocumentProcedure 1-13
1–10 PlsqlUploadAsLongRaw 1-13
1–11 PlsqlCGIEnvironmentList with Environment Variable Overrides 1-19
1–12 PlsqlCGIEnvironmentList with New Environment Variable 1-19
3–1 System Default Settings Specified with the PlsqlExclusionList Directive 3-4

List of Figures

1–1 Overview of the Process When a Server Receives a Client Request 1-2
4–1 Validation Technique 4-14
4–2 Validation Technique-Second Request 4-15
4–3 The Expires Technique 4-16
4–4 The Expires Technique-Second Request 4-17

List of Tables

1–1 Invoking mod_plsql Parameters 1-3
1–2 Two Parameter Interface Parameters 1-8
1–3 Four Parameter Interface Parameters 1-8
2–1 Installing Required Packages Parameters 2-2
3–1 Authentication Modes Used with mod_plsql 3-1
3–2 Custom Authentication Modes and Callback Functions 3-3
4–1 Database Access Descriptor (DAD) Parameters Recommended Setting Summary 4-1
4–2 Caching Options 4-2
4–3 Validation Model Parameters 4-14
4–4 Expires Model Parameters 4-16
4–5 Type of User Determined by Authentication Mode 4-18
4–6 Primary owa_cache Functions 4-18
4–7 mod_plsql cache.conf Configuration Parameter Summary 4-19

Preface

This manual describes how to install, configure, and maintain mod_plsql.

New Features

New features in this release of mod_plsql include:

- Enhanced cache clean up algorithm allowing users to configure the time to clean up the cache.
- Enhanced OWA packages to allow better byte-packing of response data from multibyte databases resulting in fewer round-trips to the database.
- Reduced PL/SQL function call overheads in heavily called OWA package APIs like `htp.p`.
- Enhanced performance logging in mod_plsql. See "What kinds of logging facilities are available in mod_plsql?" in Appendix A, "Frequently Asked Questions" for more information.
- Enhanced security by obscuring schema passwords in the DAD configuration.
- Removed the requirement to have `/pls` in the virtual path for accessing mod_plsql. It can still be used, but is no longer required. mod_plsql has been integrated in the Oracle HTTP Server configuration and so a mod_plsql URL no longer needs to start with a "`/pls`" prefix.

Audience

This document is intended for database administrators who need to install, configure, and maintain mod_plsql, to effectively secure Oracle database access.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

You may also find the following manuals useful:

- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle HTTP Server Administrator's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Understanding mod_plsql

mod_plsql provides support for building PL/SQL-based applications on the Web. PL/SQL stored procedures retrieve data from a database and generate HTTP responses containing data and code to display in a Web browser. mod_plsql also supports other Oracle products such as OracleAS Portal.

This chapter discusses the following topics:

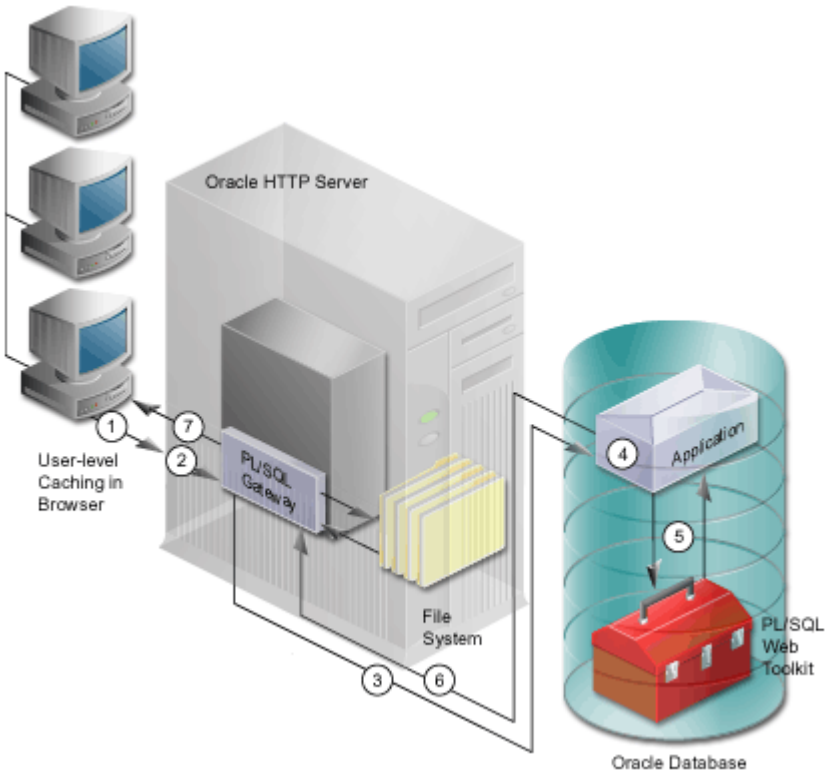
- Processing Client Requests
- Database Access Descriptors (DADs)
- Invoking mod_plsql
- Transaction Mode
- Supported Data Types
- Parameter Passing
- File Upload and Download
- Path Aliasing (Direct Access URLs)
- Common Gateway Interface (CGI) Environment Variables
- Restrictions in mod_plsql

1.1 Processing Client Requests

mod_plsql is an Oracle HTTP Server plug-in that communicates with the database. It maps browser requests into database stored procedure calls over a SQL*Net connection. It is often indicated by a `/pls` virtual path.

The following scenario provides an overview of what steps occur when a server receives a client request:

Figure 1–1 Overview of the Process When a Server Receives a Client Request



1. The Oracle HTTP Server receives a PL/SQL Server Page request from a client browser.
2. The Oracle HTTP Server routes the request to mod_plsql.
3. The request is forwarded by mod_plsql to the Oracle Database. By using the configuration information stored in your DAD, mod_plsql connects to the database.
4. mod_plsql prepares the call parameters, and invokes the PL/SQL procedure in the application.
5. The PL/SQL procedure generates an HTML page using data and the PL/SQL Web Toolkit accessed from the database.
6. The response is returned to mod_plsql.
7. The Oracle HTTP Server sends the response to the client browser.

The procedure that mod_plsql invokes returns the HTTP response to the client. To simplify this task, mod_plsql includes the PL/SQL Web Toolkit, which contains a set of packages called the owa packages. Use these packages in your stored procedure to get information about the request, construct HTML tags, and return header information to the client. Install the toolkit in a common schema so that all users can access it.

1.2 Database Access Descriptors (DADs)

Each mod_plsql request is associated with a Database Access Descriptor (DAD), a set of configuration values used for database access. A DAD specifies information such as:

- the database alias (Oracle Net service name).

- a connect string, if the database is remote.
- a procedure for uploading and downloading documents.

You can also specify username and password information in a DAD. If they are not specified, the user is prompted to enter a username and password when the URL is invoked.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for descriptions of the DAD parameters and an overview of the mod_plsql configuration files.

1.3 Invoking mod_plsql

To invoke mod_plsql in a Web browser, input the URL in the following format:

```
protocol://hostname[:port]/DAD_location/[[!][schema.][package.]proc_name[?query_string]]
```

Table 1–1 Invoking mod_plsql Parameters

Parameter	Description
protocol	Either `http` or `https`. For SSL, use `https`.
hostname	The machine where the Web server is running.
port (optional)	The port at which the Web server is listening. If omitted, port 80 is assumed.
DAD location	A virtual path to handle PL/SQL requests that you have configured in the Web server. The DAD location can contain only ASCII characters.
! character (optional)	Indicates to use the flexible parameter passing scheme. See Section 1.6.2, "Flexible Parameter Passing" for more information.
schema (optional)	The database schema name. If omitted, name resolution for *package.proc_name* occurs based on the database user that the URL request is processed as.
package (optional)	The package that contains the PL/SQL stored procedure. If omitted, the procedure is standalone.
proc_name	The PL/SQL stored procedure to run. This must be a procedure and not a function. It can accept only IN arguments.
?query_string (optional)	The parameters for the stored procedure. The string follows the format of the GET method. For example: ■ Multiple parameters are separated with the `&` character. Space characters in the values to be passed in are replaced with the + character. ■ If you use HTML forms to generate the string (as opposed to generating the string yourself), the formatting is done automatically. ■ The HTTP request may also choose the HTTP POST method to post data to mod_plsql. See "POST, GET, and HEAD Methods" on page 1-4 for more information.

Example 1–1 Invoking A Procedure That Does Not Take Arguments

```
http://www.acme.com:9000/pls/mydad/mypackage.myproc
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it passes the request to mod_plsql. This is because the `/pls/mydad` indicates that the Web server is configured to invoke mod_plsql. It then uses the DAD associated with `/pls/mydad` and runs the `myproc` procedure stored in `mypackage`.

Example 1–2 Invoking A Procedure That Takes Arguments

```
http://www.acme.com:9000/pls/mydad/mypackage.myproc?a=v&b=1
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it uses the DAD associated with `/pls/mydad` and runs the `myproc` procedure stored in `mypackage`, and passes two arguments, `a` and `b`, with the values `v`, and `1` to the procedure.

Example 1–3 Invoking the Default Procedure Stored in the DAD Configuration

```
http://www.acme.com:9000/pls/mydad
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it uses the DAD associated with `/pls/mydad` and invokes the default procedure configured in the DAD. For example, if the configuration parameter `PlsqlDefaultPage` in the DAD `/pls/mydad` is set to `myschema.mypackage.myproc`, then the procedure `myschema.mypackage.myproc` is invoked for the request.

In this example, the default home page for the `mydad` DAD (as specified in the DAD Configuration) is displayed.

POST, GET, and HEAD Methods

The POST, GET, and HEAD methods in the HTTP protocol instruct browsers on how to pass parameter data (usually in the form of name-value pairs) to applications. The parameter data is generated by HTML forms.

mod_plsql applications can use any of the methods. Each method is as secure as the underlying transport protocol (HTTP or HTTPS).

- When using the POST method, parameters are passed in the request body. Generally, if you are passing large amounts of parameter data to the server, use the POST method.
- When using the GET method, parameters are passed using a query string. The limitation of this method is that the length of the value in a name-value pair cannot exceed the maximum length for the value of an environment variable, as imposed by the underlying operating system. In addition, operating systems have a limit on how many environment variables you can define.
- When using the HEAD method, it has the same functionality as the GET method. The only difference is that only the HTTP status line and the HTTP headers are passed back. No content data is streamed back to the browser. This is useful for monitoring tools in which you are only interested if the request is processed correctly.
- Mixed Mode - In mod_plsql you can pass some of the parameters in a query string and the remaining ones as POST data. For example, if you have a procedure `foo` (a varchar2, b number), and want to pass values "v" and "1" to 'a' and 'b' respectively, you could do so in three ways to create URLs:
  - All values are specified as part of the query string.

```
http://host:port/pls/DAD/foo?a=v&b=1
```

– All values are specified as part of the POST data.

```
http://host:port/pls/DAD/foo, POST data="a=v&b=1"
```

– Some of the parameters are specified in the URL and the rest in the POST data.

```
http://host:port/pls/DAD/foo?a=v, POST data="b=1"
```

Note: POST data is generated as part of the input fields on a HTML form. You should not create the POST string manually in the PL/SQL procedure, or in the URL. The Submit operation of the HTML form will generate a POST request and pass the value to your procedure.

1.4 Transaction Mode

After processing a URL request for a procedure invocation, mod_plsql performs a rollback if there were any errors. Otherwise, it performs a commit. This mechanism does not allow a transaction to span across multiple HTTP requests. In this stateless model, applications typically maintain state using HTTP cookies or database tables.

1.5 Supported Data Types

Because HTTP supports character streams only, mod_plsql supports the following subset of PL/SQL data types:

- NUMBER
- VARCHAR2
- TABLE OF NUMBER
- TABLE OF VARCHAR2

Records are not supported.

1.6 Parameter Passing

mod_plsql supports:

- Parameter passing by name

  Each parameter in a URL that invokes procedure or functions identified by a unique name. Overloaded parameters are supported. See Section 1.6.1, "Parameter Passing by Name (Overloaded Parameters)" for more information.

- Flexible parameter passing

  Procedures are prefixed by a ! character. See Section 1.6.2, "Flexible Parameter Passing" for more information.

- Large (up to 32K) parameters passing

  See Section 1.6.3, "Large Parameter Passing" for more information.

Note: mod_plsql handles multi-value variables by storing the values in a PL/SQL table. This enables you to be flexible about how many values the user can pick, and it makes it easy for you to process the user's selections as a unit. Each value is stored in a row in the PL/SQL table, starting at index 1. The first value (in the order that it appears in the query string) of a variable that has multiple values is placed at index 1, the second value of the same variable is placed at index 2, and so on. If the order of the values in the PL/SQL table is significant in your procedure, you need to determine the order in which the variables appear in the query string, or modify your PL/SQL application to do the ordering internally.

If you do not have variables with multiple values, the order in which the variables appear does not matter, because their values are passed to the procedure's parameters by name, and not by position.

The PL/SQL tables used as parameters in the mod_plsql environment must have a base type of `VARCHAR2`. Oracle can convert `VARCHAR2` to other data types such as `NUMBER`, `DATE`, or `LONG`. The maximum length of a `VARCHAR2` variable is 32K.

If you cannot guarantee that at least one value will be submitted to the PL/SQL table (for example, the user can select no options), use a hidden form element to provide the first value. Not providing a value for the PL/SQL table produces an error, and you cannot provide a default value for a PL/SQL table.

1.6.1 Parameter Passing by Name (Overloaded Parameters)

Overloading allows multiple subprograms (procedures or functions) to have the same name, but differ in the number, order, or the datatype family of the parameters. When you call an overloaded subprogram, the PL/SQL compiler determines which subprogram to call based on the data types passed.

PL/SQL enables you to overload local or packaged subprograms. Standalone subprograms cannot be overloaded.

You must give parameters different names for overloaded subprograms that have the same number of parameters. Because HTML data is not associated with datatypes, mod_plsql does not know which version of the subprogram to call.

For example, although PL/SQL enables you to define two procedures using the same parameter names for the procedures, an error occurs if you use this with mod_plsql.

```
-- legal PL/SQL, but not for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2);
  PROCEDURE my_proc (val IN NUMBER);
END my_pkg;
```

To avoid the error, name the parameters differently. For example:

```
-- legal PL/SQL and also works for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (valvc2 IN VARCHAR2);
  PROCEDURE my_proc (valnum IN NUMBER);
END my_pkg;
```

The URL to invoke the first version of the procedure looks similar to:

```
http://www.acme.com/pls/mydad/my_pkg.my_proc?valvc2=input
```

The URL to invoke the second version of the procedure looks similar to:

```
http://www.acme.com/pls/mydad/my_pkg.my_proc?valnum=34
```

Overloading and PL/SQL Arrays

If you have overloaded PL/SQL procedures where the parameter names are identical, but the data type is *owa_util.ident_arr* (a table of varchar2) for one procedure and a scalar type for another procedure, mod_plsql can still distinguish between the two procedures. For example, if you have the following procedures:

```
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2); -- scalar data type
  PROCEDURE my_proc (val IN owa_util.ident_arr); -- array data type
END my_pkg;
```

Each of these procedures has a single parameter of the same name, `val`.

When mod_plsql gets a request that has only one value for the `val` parameter, it invokes the procedure with the scalar data type.

Example 1–4 Sending a URL to Execute the Scalar Version of a Procedure

Send the following URL to execute the scalar version of the procedure:

```
http://www.acme.com/pls/mydad/my_proc?val=john
```

When mod_plsql gets a request with more than one value for the `val` parameter, it then invokes the procedure with the array data type.

Example 1–5 Sending a URL to Execute the Array Version of a Procedure

Send the following URL to execute the array version of the procedure:

```
http://www.acme.com/pls/mydad/my_proc?val=john&val=sally
```

To ensure that the array version executes, use hidden form elements on your HTML page to send dummy values that are checked and discarded in your procedure.

1.6.2 Flexible Parameter Passing

You can have HTML forms from which users can select any number of elements. If these elements have different names, you would have to create overloaded procedures to handle each possible combination. Alternatively, you could insert hidden form elements to ensure that the names in the query string are consistent each time, regardless of what elements the user chooses. mod_plsql makes this operation easier by supporting flexible parameter passing to handle HTML forms where users can select any number of elements.

To use flexible parameter passing for a URL-based procedure invocation, prefix the procedure with an exclamation mark (!) in the URL. You can use two or four parameters. The two parameter interface provides improved performance with mod_plsql. The four parameter interface is supported for compatibility.

1.6.2.1 Two Parameter Interface

```
procedure [proc_name]
```

```
     (name_array IN [array_type],
     value_array IN  [array_type]);
```

Table 1–2 Two Parameter Interface Parameters

Parameter	Description
proc_name (required)	The name of the PL/SQL procedure that you are invoking.
name_array	The names from the query string (indexed from 1) in the order submitted.
value_array	The values from the query string (indexed from 1) in the order submitted.
array_type (required)	Any PL/SQL index-by table of varchar2 type (Example, owa.vc_arr).

Example 1–6 Two Parameter Interface

If you send the following URL:

```
http://www.acme.com/pls/mydad/!scott.my_proc?x=john&y=10&z=doe
```

The exclamation mark prefix (!) instructs mod_plsql to use flexible parameter passing. It invokes procedure *scott.myproc* and passes it the following two arguments:

```
name_array ==> ('x', 'y', 'z')
value_array ==> ('john', '10', 'doe')
```

Note: When using this style of Flexible Parameter Passing, the procedure must be defined with the parameters *name_array* and *value_array*. The datatypes of these arguments should match the datatypes shown in the example.

1.6.2.2 Four Parameter Interface

The four parameter interface is supported for compatibility.

```
procedure [proc_name]
     (num_entires IN NUMBER,
     name_array  IN  [array_type],
     value_array IN  [array_type],
     reserved in [array_type]);
```

Table 1–3 Four Parameter Interface Parameters

Parameter	Description
proc_name (required)	The name of the PL/SQL procedure that you are invoking.
num_entries	The number of name_value pairs in the query string
name_array	The names from the query string (indexed from 1) in the order submitted.
value_array	The values from the query string (indexed from 1) in the order submitted.
reserved	Not used. It is reserved for future use.

Table 1–3 (Cont.) Four Parameter Interface Parameters

Parameter	Description
array_type (required)	Any PL/SQL index-by table of varchar2 type (Example, owa.vc_arr).

Example 1–7 Four Parameter Interface

If you send the following URL, where the query_string has duplicate occurrences of the name "x":

```
http://www.acme.com/pls/mydad/!scott.my_pkg.my_proc?x=a&y=b&x=c
```

The exclamation mark prefix (!) instructs mod_plsql to use flexible parameter passing. It invokes procedure `scott.my_pkg.myproc` and passes it the following arguments:

```
num_entries ==> 3
name_array ==> ('x', 'y', 'x');
value_array ==> ('a', 'b', 'c')
reserved ==> ()
```

Note: When using this style of Flexible Parameter Passing, the procedure must be defined with the parameters *num_entries*, *name_array*, *value_array*, and *reserved*. The datatypes of these arguments should match the datatypes shown in the example.

1.6.3 Large Parameter Passing

The values passed as scalar arguments and the values passed as elements to the index-by table of varchar2 arguments can be up to 32K in size.

For example, when using flexible parameter passing (described in Section 1.6.2, "Flexible Parameter Passing"), each name or value in the *query_string* portion of the URL gets passed as an element of the `name_array` or `value_array` argument to the procedure being invoked. These names or values can be up to 32KB in size.

1.7 File Upload and Download

mod_plsql enables you to:

- Upload and download files as raw byte streams without any character set conversions. The files are uploaded into the document table. A primary key is passed to the PL/SQL upload handler routine so that it can retrieve the appropriate table row.
- Specify one or more tables for each application, for uploaded files so that files from different applications are not mixed together.
- Provide access to files in these tables through a URL format that doesn't use query strings, for example:

  ```
  http://www.acme.com:9000/pls/mydad/docs/cs250/lecture1.htm
  ```

  This is required to support uploading a set of files that have relative URL references to each other.
- Upload multiple files for each form submission.

- Upload files into LONG RAW and BLOB (Binary Large Object) types of columns in the document table.

This section discusses the following:

- Document Table Definition
- Old Style Document Table Definition
- Configuration Parameters for Document Upload/Downloading
- File Upload
- Specifying Attributes (Mime Types) of Uploaded Files
- Uploading Multiple Files
- File Download
- Direct BLOB Download

1.7.1 Document Table Definition

You can specify the document storage table for each DAD. The document storage table must have the following definition:

```
CREATE TABLE [table_name] (
     NAME           VARCHAR2(256) UNIQUE NOT NULL,
     MIME_TYPE      VARCHAR2(128),
     DOC_SIZE       NUMBER,
     DAD_CHARSET    VARCHAR2(128),
     LAST_UPDATED   DATE,
     CONTENT_TYPE   VARCHAR2(128),
     [content_column_name] [content_column_type]
     [ , [content_column_name] [content_column_type]]
);
```

Users can choose the `table_name`. The `content_column_type` type must be either LONG RAW or BLOB.

The `content_column_name` depends on the corresponding `content_column_type`:

- If the content_column_type is LONG RAW, the `content_column_name` must be CONTENT.
- If the content_column_type is BLOB, the `content_column_name` must be BLOB_CONTENT.

An example of legal document table definition is:

```
CREATE TABLE MYDOCTABLE (
  NAME               VARCHAR(128)   UNIQUE NOT NULL,
  MIME_TYPE          VARCHAR(128),
  DOC_SIZE           NUMBER,
  DAD_CHARSET        VARCHAR(128),
  LAST_UPDATED       DATE,
  CONTENT_TYPE       VARCHAR(128),
  CONTENT            LONG RAW,
  BLOB_CONTENT       BLOB ;
);
```

1.7.1.1 Semantics of the CONTENT Column

The contents of the table are stored in a content column. There can be more than one content column in a document table. However, for each row in the document table, only one of the content columns is used. The other content columns are set to NULL.

1.7.1.2 Semantics of the CONTENT_TYPE Column

The `content_type` column tracks in which content column the document is stored. When a document is uploaded, mod_plsql sets the value of this column to the type name.

For example, if a document was uploaded into the BLOB_CONTENT column, then the `CONTENT_TYPE` column for the document is set to the string 'BLOB'.

1.7.1.3 Semantics of the LAST_UPDATED Column

The `LAST_UPDATED` column reflects a document's creation or last modified time. When a document is uploaded, mod_plsql sets the `LAST_UPDATED` column for the document to the database server time.

If an application then modifies the contents or attributes of the document, it must also update the `LAST_UPDATED` time.

mod_plsql uses the `LAST_UPDATED` column to check and indicate to the HTTP client (browser) if the browser can use a previously cached version of the document. This reduces network traffic and improves server performance.

1.7.1.4 Semantics of the DAD_CHARSET Column

The `DAD_CHARSET` column keeps track of the character set setting at the time of the file upload. This column is reserved for future use.

1.7.2 Old Style Document Table Definition

For backward capability with the document model used by older releases of WebDB 2.x, mod_plsql also supports the following old definition of the document storage table where the CONTENT_TYPE, DAD_CHARSET and LAST_UPDATED columns are not present.

```
/* older style document table definition (DEPRECATED) */
CREATE TABLE [table_name]
(
    NAME         VARCHAR2(128),
    MIME_TYPE    VARCHAR2(128),
    DOC_SIZE     NUMBER,
    CONTENT      LONG RAW
);
```

1.7.3 Configuration Parameters for Document Upload/Downloading

The following configuration parameters in the DAD affect a document upload/download operation:

- "PlsqlDocumentTablename"
- "PlsqlDocumentPath (Document Access Path)"
- "PlsqlDocumentProcedure (Document Access Procedure)"
- "PlsqlUploadAsLongRaw"

Example 1–8 Parameters for Document Upload/Download

If the configuration for these parameters in a DAD is as follows:

```
PlsqlDocumentTablename   scott.my_document_table
PlsqlUploadAsLongRaw   html
PlsqlDocumentPath   docs
PlsqlDocumentProcedure   scott.my_doc_download_procedure
```

then:

- mod_plsql will retrieve data from, or store to a database table called **my_document_table** in the **scott** schema.
- All file extensions except `.html` will be uploaded to the document table as BLOBs. All files with `.html` extension will be uploaded as *Long Raw*.
- All URLs which have the keyword **docs** immediately following the DAD location will result in invocation of the procedure **scott.my_doc_download_procedure**.

  Typically, this procedure will call **wpg_docload.download_file** to initiate a file download for a file whose name is based on the URL specification.

A simple example with the preceding configuration is:

```
http://www.acme.com/pls/dad/docs/index.html
```

This results in downloading of the file `index.html` from the *Long Raw* column of the database table **scott.my_document_table**. Note that the application procedure has full control on the file download to initiate, and has the flexibility to define a more complex `PlsqlDocumentProcedure` that implements file-level access controls and versioning.

Note: The application defined procedure **scott.my_doc_download_procedure** has to be defined without arguments, and should rely on the CGI environment variables to process the request.

1.7.3.1 PlsqlDocumentTablename

The `PlsqlDocumentTablename` parameter specifies the table for storing documents when file uploads are performed through this DAD.

Syntax:

```
PlsqlDocumentTablename  [document_table_name]

PlsqlDocumentTablename  my_documents
```

or,

```
PlsqlDocumentTablename  scott.my_document_table
```

1.7.3.2 PlsqlDocumentPath (Document Access Path)

The `PlsqlDocumentPath` parameter specifies the path element to access a document. The `PlsqlDocumentPath` parameter follows the DAD name in the URL. For example, if the document access path is `docs`, then the URL would look similar to:

```
http://www.acme.com/pls/mydad/docs/myfile.htm
```

The `mydad` is the DAD name and `myfile.htm` is the file name.

Syntax:

```
PlsqlDocumentPath   [document_access_path_name]
```

1.7.3.3 PlsqlDocumentProcedure (Document Access Procedure)

The `PlsqlDocumentProcedure` procedure is an application-specified procedure. It has no parameters and processes a URL request with the document access path. The document access procedure calls `wpg_docload.download_file(filename)` to download a file. It knows the filename based on the URL specification. For example, an application can use this to implement file-level access controls and versioning. An example of this is in Section 1.7.7, "File Download".

Syntax:

```
PlsqlDocumentProcedure   [document_access_procedure_name]
```

Example 1–9 PlsqlDocumentProcedure

```
PlsqlDocumentProcedure   my_access_procedure
```

or,

```
PlsqlDocumentProcedure   scott.my_pkg.my_access_procedure
```

1.7.3.4 PlsqlUploadAsLongRaw

The DAD parameter, `PlsqlUploadAsLongRaw`, configures file uploads based on their file extensions. The value of a `PlsqlUploadAsLongRaw` DAD parameter is a *one-entry-for-each-line* list of file extensions. Files with these extensions are uploaded by mod_plsql into the content column of `LONG RAW` type in the document table. Files with other extensions are uploaded into the BLOB content column.

The file extensions can be text literals (jpeg, gif, and so on) or an asterisk (*) matches any file whose extension has not been listed in the `PlsqlUploadAsLongRaw` setting.

Syntax:

```
PlsqlUploadAsLongRaw   [file_extension]
PlsqlUploadAsLongRaw *
```

`[file_extension]` is an extension for a file (with or without the '.' character, for example, 'txt' or '.txt') or the wildcard character *.

Example 1–10 PlsqlUploadAsLongRaw

```
PlsqlUploadAsLongRaw   html
PlsqlUploadAsLongRaw   txt
PlsqlUploadAsLongRaw   *
```

1.7.4 File Upload

To send files from a client machine to a database, create an HTML page that contains:

- A FORM tag whose *enctype* attribute is set to `multipart/form-data` and whose *action* attribute is associated with a mod_plsql procedure call, referred to as the "action procedure."
- An INPUT element whose type and name attributes are set to file. The `INPUT type="file"` element enables a user to browse and select files from the file system.

When a user clicks **Submit**, the following events occur:

1. The browser uploads the file specified by the user as well as other form data to the server.
2. mod_plsql stores the file contents in the database in the document storage table. The table name is derived from the `PlsqlDocumentTablename` DAD setting.
3. The action procedure specified in the *action* attribute of the FORM is run (similar to invoking a mod_plsql procedure without file upload).

> **Note:** The parsing of HTML documents is deprecated in mod_plsql. mod_plsql used to parse the content of an HTML file when it was uploaded, and identified other files that the HTML document was referring to. This information was then stored into a table. The table name was constructed by appending the name of the document table with "part". This functionality was found to be not of use to customers and has been deprecated, starting in version 9.0.4 of mod_plsql.

The following example shows an HTML form that lets a user select a file from the file system to upload. The form contains other fields to provide information about the file.

```
<html>
   <head>
      <title>test upload</title>
   </head>
   <body>
   <FORM enctype="multipart/form-data"
      action="pls/mydad/write_info"
      method="POST">
      <p>Author's Name:<INPUT type="text" name="who">
      <p>Description:<INPUT type="text" name="description"><br>
      <p>File to upload:<INPUT type="file" name="file"><br>
      <p><INPUT type="submit">
   </FORM>
   </body>
</html>
```

When a user clicks **Submit** on the form:

1. The browser uploads the file listed in the `INPUT type="file"` element.
2. The `write_info` procedure then runs.
3. The procedure writes information from the form fields to a table in the database and returns a page to the user.

> **Note:** The action procedure does not have to return anything to the user, but it is a good idea to let the user know whether the Submit succeeded or failed, as shown subsequently.

```
procedure write_info (
     who         in varchar2,
     description in varchar2,
     file        in varchar2) as
begin
     insert into myTable values (who, description, file);
     htp.htmlopen;
     htp.headopen;
```

```
        htp.title('File Uploaded');
        htp.headclose;
        htp.bodyopen;
        htp.header(1, 'Upload Status');
        htp.print('Uploaded ' || file || ' successfully');
        htp.bodyclose;
        htp.htmlclose;
    end;
```

The filename obtained from the browser is prefixed with a generated directory name to reduce the possibility of name conflicts. The "action procedure" specified in the form renames this name. So, for example, when `/private/minutes.txt` is uploaded, the name stored in the table by the mod_plsql is `F9080/private/minutes.txt`. The application can rename this in the called stored procedure. For example, the application can rename it to `scott/minutes.txt`.

1.7.5 Specifying Attributes (Mime Types) of Uploaded Files

In addition to renaming the uploaded file, the stored procedure can alter other file attributes. For example, the form in the example from Section 1.7.4, "File Upload" could display a field for allowing the user to input the uploaded document's Multipurpose Internet Mail Extension (MIME) type.

The MIME type can be received as a parameter in `write_info`. The document table would then store the mime type for the document instead of the default mime type that is parsed from the multipart form by mod_plsql when uploading the file.

1.7.6 Uploading Multiple Files

To send multiple files in a single submit, the upload form must include multiple <INPUT type="file" name="file"> elements. If more than one file INPUT element defines `name` to be of the same name, then the action procedure must declare that parameter name to be of type owa.vc_arr. The names defined in the file INPUT elements could also be unique, in which case, the action procedure must declare each of them to be of varchar2. For example, if a form contained the following elements:

```
<INPUT type="file" name="textfiles">
<INPUT type="file" name="textfiles">
<INPUT type="file" name="binaryfile">
```

As a result, the action procedure must contain the following parameters:

```
procedure handle_text_and_binary_files(textfiles IN owa.vc_arr, binaryfile IN
varchar2).
```

1.7.7 File Download

After you have sent files to the database, you can download them, delete them from the database, and read and write their attributes.

To download a file, create a stored procedure without parameters that calls `wpg_docload.download_file` (file_name) to initiate the download.

The HTML page presented to the user simply has a link to a URL, which includes the Document Access Path and specifies the file to be downloaded.

For example, if the DAD specifies that the Document Access Path is `docs` and the Document Access Procedure is `mydad.process_download`, then the `mydad.process_download` procedure is called when the user clicks on the URL:

```
http://www.acme.com:9000/pls/mydad/docs/myfile.htm
```

An example implementation of process_download is:

```
procedure process_download is
v_filename varchar2(255);
begin
  -- getfilepath() uses the SCRIPT_NAME and PATH_INFO cgi
  -- environment variables to construct the full path name of
  -- the file URL, and then returns the part of the path name
  -- following '/docs/'
  v_filename := getfilepath;
  select name into v_filename from plsql_gateway_doc
  where UPPER(name) = UPPER(v_filename);
  -- now we call docload.download_file to initiate
  -- the download.
  wpg_docload.download_file(v_filename);
exception
  when others then
v_filename := null;
end process_download;
```

Any time you call `wpg_docload.download_file(filename)` from a procedure running in mod_plsql, a download of the file *`filename`* is initiated. However, when a file download begins, no other HTML (produced through HTP interfaces) generated by the procedure, is passed back to the browser.

mod_plsql looks for the filename in the document table. There must be a unique row in the document table whose NAME column matches the filename. mod_plsql generates the HTTP response headers based on the information in the MIME_TYPE column of the document table. The `content_type` column's value determines which content columns the document's content comes from. The contents of the document are sent as the body of the HTTP response.

1.7.8 Direct BLOB Download

You can also download contents stored as Binary Large Object (BLOB) data type.

1. Create a stored procedure that calls wpg_docload.download_file(blob) where blob is of data type BLOB. Since mod_plsql has no information about the contents in the BLOB, you must supply them.
2. Setup the Content-Type and other headers.

   Example: The following procedure uses the name from the argument to select a BLOB from a table and initiates the Direct BLOB download:

   ```
   create or replace procedure download_blob(varchar2 name) is
   myblob blob;
   begin
   ```

   a. Select the BLOB out of mytable using the name argument

      ```
      select blob_data into myblob from mytable where blob_name = name;
      ```

   b. Setup headers which describes the content

      ```
      owa_util.mime_header('text/html', FALSE);
      htp.p('Content-Length: ' || dbms_lob.getlength(myblob));
      owa_util.http_header_close;
      ```

   c. Initiate Direct BLOB download

```
wpg_docload.download_file(myblob);
end;
```

The structure of the `mytable` table:

```
create table mytable
(
blob_name varchar2(128),
blob_data blob
);
```

3. The HTML page presented to the user has a link to a URL that calls this stored procedure with the correct argument(s).
4. When a Direct BLOB download is initiated, no other HTML (produced through the HTP interface) generated by the procedure is passed back to the browser.

1.8 Path Aliasing (Direct Access URLs)

Path Aliasing enables applications using mod_plsql to provide direct reference to its objects using simple URLs. The Path Aliasing functionality is a generalization of how the document download functionality is provided. The following configuration parameters in the DAD are used for Path Aliasing:

- PlsqlPathAlias
- PlsqlPathAliasProcedure

For Example, if the configuration for these parameters in a DAD is as follows:

```
PlsqlPathAlias   myalias
PlsqlPathAliasProcedure   scott.my_path_alias_procedure
```

then, all URLs that have the keyword **myalias** immediately following the DAD location will invoke the procedure **scott.my_path_alias_procedure**. Based on the URL specification, this procedure can initiate an appropriate response.

Note: The application defined procedure **scott.my_path_alias_procedure** has to be defined to take one argument of type *varchar2* called **p_path**. This argument will receive everything following the keyword used in *PlsqlPathAlias*.

For example, in the preceding configuration, the URL:

```
http://www.acme.com/pls/dad/myalias/MyFolder/MyItem
```

will result in the procedure **scott.my_path_alias_procedure** receiving the argument **MyFolder/MyItem**.

1.9 Common Gateway Interface (CGI) Environment Variables

The OWA_UTIL package provides an API to get the values of CGI environment variables. The variables provide context to the procedure being executed through mod_plsql. Although mod_plsql is not operated through CGI, the PL/SQL application invoked from mod_plsql can access these CGI environment variables.

The list of CGI environment variables is as follows:

- HTTP_AUTHORIZATION
- DAD_NAME

- DOC_ACCESS_PATH
- HTTP_ACCEPT
- HTTP_ACCEPT_CHARSET
- HTTP_ACCEPT_LANGUAGE
- HTTP_COOKIE
- HTTP_HOST
- HTTP_PRAGMA
- HTTP_REFERER
- HTTP_USER_AGENT
- PATH_ALIAS
- PATH_INFO
- HTTP_ORACLE_ECID
- DOCUMENT_TABLE
- REMOTE_ADDR
- REMOTE_HOST
- REMOTE_USER
- REQUEST_CHARSET (refer to Section 1.9.2.1, "REQUEST_CHARSET CGI Environment Variable")
- REQUEST_IANA_CHARSET (refer to Section 1.9.2.2, "REQUEST_IANA_CHARSET CGI Environment Variable")
- REQUEST_METHOD
- REQUEST_PROTOCOL
- SCRIPT_NAME
- SCRIPT_PREFIX
- SERVER_NAME
- SERVER_PORT
- SERVER_PROTOCOL

A PL/SQL application can get the value of a CGI environment variable using the owa_util.get_cgi_env interface.

Syntax:

```
owa_util.get_cgi_env(param_name in varchar2) return varchar2;
```

param_name is the name of the CGI environment variable. param_name is case-insensitive.

1.9.1 Adding and Overriding CGI Environment Variables

The `PlsqlCGIEnvironmentList` DAD parameter is a *one-entry-for-each-line* list of name and value pairs that can override any environment variables or add new ones. If the name is one of the original environment variables (as listed in Section 1.9, "Common Gateway Interface (CGI) Environment Variables"), that environment variable is overridden with the given value. If the name is not in the original list, a

new environment variable is added into the list with that same name and value given in the parameter.

> **Note:** Refer to the *Oracle HTTP Server Administrator's Guide* for information about the mod_plsql Configuration Files.

If no value is specified for the parameter, then the value is obtained from the Oracle HTTP Server. With Oracle HTTP Server, you can pass the DOCUMENT_ROOT CGI Environment variable by specifying:

```
PlsqlCGIEnvironmentList DOCUMENT_ROOT
```

New environment variables passed in through this configuration parameter are available to the PL/SQL application through the owa_util.get_cgi_env interface.

Example 1–11 PlsqlCGIEnvironmentList with Environment Variable Overrides

```
PlsqlCGIEnvironmentList SERVER_NAME=myhost.mycompany.com
PlsqlCGIEnvironmentList
REMOTE_USER=testuser
```

This example overrides the SERVER_NAME and the REMOTE_USER CGI environment variables with the given values since they are part of the original list.

Example 1–12 PlsqlCGIEnvironmentList with New Environment Variable

```
PlsqlCGIEnvironmentList MYENV_VAR=testing
PlsqlCGIEnvironmentList SERVER_NAME=
PlsqlCGIEnvironmentList
REMOTE_USER=user2
```

This example overrides the SERVER_NAME and the REMOTE_USER variables. The SERVER_NAME variable is deleted since there is no value given to it. A new environment variable called MYENV_VAR is added since it is not part of the original list. It is assigned the value of "testing".

1.9.2 PlsqlNLSLanguage

For mod_plsql, the Globalization Support variable (`PlsqlNLSLanguage`) can be set either as an environment variable or at the DAD level, the `PlsqlNLSLanguage` parameter of the database must match that of the Oracle HTTP Server, or the `PlsqlNLSLanguage` parameter of the database and Oracle HTTP Server, must be of fixed character width and both must be the same size.

If `PlsqlNLSLanguage` is not configured at the DAD level, the Globalization Support setting is picked up from the environment. If it does not exist, the default rules apply for NLS_LANG settings for Oracle.

1.9.2.1 REQUEST_CHARSET CGI Environment Variable

Every request to mod_plsql is associated with a DAD. The CGI environment variable REQUEST_CHARSET is set as follows:

- The REQUEST_CHARSET is set to the default character set in use, derived from the `PlsqlNLSLanguage` environment variable. However, if the DAD level `PlsqlNLSLanguage` parameter is set, that derives the character set information instead.

The PL/SQL application can access this information by a function call of the form:

```
owa_util.get_cgi_env('REQUEST_CHARSET');
```

1.9.2.2 REQUEST_IANA_CHARSET CGI Environment Variable

This is the IANA (Internet Assigned Number Authority) equivalent of the REQUEST_CHARSET CGI environment variable. IANA is an authority that globally coordinates the standards for charsets on the Internet.

1.10 Restrictions in mod_plsql

The following restrictions exist in mod_plsql:

- The maximum length of the HTTP cookie header is 32000 bytes. Values higher than this generate an error. This limit is due to the PL/SQL varchar2 limit.
- The maximum length of any single cookie within the HTTP cookie is 3990. Values higher than this generate an error. This limit is due to the OCI array bind limit of strings in arrays.
- There is a hard maximum cookie limit in mod_plsql that limits the number of cookies being set at any given time. That limit is set to 20. Anything over 20 will be dropped.
- The PL/SQL Gateway does not support calling procedures with OUT parameters to be called from a Web interface. Doing this may result in ORA-6502 errors. The recommended approach is not to call any procedure that has OUT variables in it. However, the current architecture will let you modify a value as long as the modified value does not exceed the length that was passed in. Existing applications that encounter this problem need to be modified in one of the following ways:
  - Implement wrappers for procedures with OUT parameters so that such procedures are not invoked directly through a browser URL.
  - Create a local variable that gets assigned the value of the parameter being passed in, and is then used for all internal changes.
- The total number of name value pairs that can be passed to a PL/SQL procedure is 2000.
- mod_plsql limits the total number of parameters that can be passed to a single procedure to 2000.
- mod_plsql limits the size of a single parameter that can be passed to a procedure to 32000 bytes.
- It is not possible to use identical DAD locations in different virtual hosts.

2

Configuring mod_plsql

This chapter describes how you can set up and use mod_plsql. It contains the following sections:

- Verifying Requirements
- Installing Required Packages
- Creating a DAD for Your PL/SQL Application

2.1 Verifying Requirements

Before you run mod_plsql, you must satisfy the following requirements:

- You must have a SYS user password on the database where you plan to load PL/SQL Web Toolkit packages required by mod_plsql.
- The database to which you plan to connect mod_plsql must be up and running.
- Oracle HTTP Server mod_plsql ships with OWA package version 9.0.4.0.1. It is recommended that the OWA packages installed in the database are at least version 9.0.4.0.1.

2.2 Installing Required Packages

After installation, if you need to use Oracle HTTP Server mod_plsql with a database that is not shipped with the product, you must manually install additional required packages using the `owaload.sql` script.

> **Note:** Even if a full database export is made with the Export utility you still must reinstall mod_plsql in the new target instance by running the `OWALOAD.SQL` script as SYS. Objects in SYS are not imported with the Import/Export mechanism, and the PL/SQL toolkit has to be installed in SYS.

1. Navigate to the directory where the `owaload.sql` file is located. This directory is *ORACLE_HOME*`/Apache/modpsql/owa.`
2. Using SQL*Plus, login to the Oracle database as the SYS user.
3. You can check the version of the OWA packages currently installed by running the following query:

```
select owa_util.get_version from dual;
```

- If the query succeeds, but shows a version less than 9.0.4.0.1, it is recommended that you install the newer OWA packages.
- If the query fails, you either do not have the OWA packages installed, or are running a very old version of OWA packages, and it is recommended that you install, or upgrade to the new OWA packages.

Note: To detect older OWA packages, see "How do I detect and clean up duplicate OWA packages installed in the database?" in Appendix A, "Frequently Asked Questions".

---

4. At an SQL prompt, run the following command:

```
@owaload.sql log_file
```

Table 2–1 Installing Required Packages Parameters

Elements	Description
owaload.sql	Installs the PL/SQL Web Toolkit packages into the SYS schema. It also creates public synonyms and makes the packages public so that all users in the database have access to them. Therefore, only one installation for each database is needed.
log_file	The installation log file. Make sure that you have write permissions to create the log file

5. Scan the log file for any errors.

Note: The `owaload` script checks the existing version of the OWA packages in the database and installs a new version only if:

- No OWA package exists or,
- Older OWA packages were detected. If your database already has the latest OWA packages or has a newer version installed, the `owaload` script does nothing and reports this in the log file.

---

6. Do a manual recompile.

Note: Installing the OWA packages invalidates all dependent objects. These packages automatically recompile on first access, but a manual recompile is recommended after the reinstallation.

After the install, check the version of the OWA packages by running "`Select owa_util.get_version from dual;`". Confirm that the version shown is 9.0.4.0.1 or later.

7. Note that public access is now granted to:
   - OWA_CUSTOM
   - OWA
   - HTF

- HTP
- OWA_COOKIE
- OWA_IMAGE
- OWA_OPT_LOCK
- OWA_PATTERN
- OWA_SEC
- OWA_TEXT
- OWA_UTIL
- OWA_CACHE
- WPG_DOCLOAD

8. Note also that the following public synonyms are created:
   - OWA_CUSTOM for OWA_CUSTOM
   - OWA_GLOBAL for OWA_CUSTOM
   - OWA for OWA
   - HTF for HTF
   - HTP for HTP
   - OWA_COOKIE for OWA_COOKIE
   - OWA_IMAGE for OWA_IMAGE
   - OWA_OPT_LOCK for OWA_OPT_LOCK
   - OWA_PATTERN for OWA_PATTERN
   - OWA_SEC for OWA_SEC
   - OWA_TEXT for OWA_TEXT
   - OWA_UTIL for OWA_UTIL
   - OWA_INIT for OWA_CUSTOM
   - OWA_CACHE for OWA_CACHE
   - WPG_DOCLOAD for WPG_DOCLOAD

2.3 Creating a DAD for Your PL/SQL Application

To access a Web-enabled PL/SQL application, you must first configure a PL/SQL Database Access Descriptor (DAD) for mod_plsql. A DAD is a set of values that specifies how mod_plsql connects to a database server to fulfill an HTTP request. Besides the connection details, a DAD contains important configuration parameters for various operations in the database, and for mod_plsql in general. For detailed instructions, refer to the mod_plsql section in the *Oracle HTTP Server Administrator's Guide*.

3

Securing Application Database Access Through mod_plsql

This chapter describes how to set up the database and PL/SQL to avoid known security problems. It covers the following topics:

- Authenticating Users Through mod_plsql
- Deauthenticating Users
- Protecting the PL/SQL Procedures Granted to PUBLIC

> **See Also:** For more information about mod_plsql configuration parameters, refer to the *Oracle HTTP Server Administrator's Guide*.

3.1 Authenticating Users Through mod_plsql

mod_plsql provides different levels of authentication in addition to those provided by the Oracle HTTP Server. The Oracle HTTP Server protects documents, virtual paths and so forth, while mod_plsql protects users logging into the database or running a PL/SQL Web application.

You can enable different authentication modes, as described in Table 3–1.

Table 3–1 Authentication Modes Used with mod_plsql

Authentication Mode	Approach
Basic	Authentication is performed using basic HTTP authentication. Most applications use basic authentication.
Global OWA	Authentication is performed using the owa_custom.authorize procedure in the schema containing the PL/SQL Web Toolkit packages.
Custom OWA	Authentication is performed using packages and procedures in the user's schema (owa_customize.authorize), or if not found, in the schema containing the PL/SQL Web Toolkit packages.
PerPackage	Authentication is performed using packages and procedures in the user's schema (packageName.authorize).
Single Sign-On	Authentication is performed using Oracle Application Server Single Sign-On. Use this mode only if your application works with OracleAS Single Sign-On.

3.1.1 Basic (Database Controlled Authentication)

The module, mod_plsql, supports authentication at the database level. It uses HTTP basic authentication but authenticates credentials by using them to attempt to log on to the database. Authentication is verified against a user database account, using user names and passwords that are either:

- stored in the DAD. The end user is not required to log in. This method is useful for Web pages that provide public information.
- provided by the users by means of the browser's Basic HTTP Authentication dialog box. The user must provide a user name and password in the dialog box.

3.1.2 Oracle HTTP Server mod_plsql Basic Authentication Mode

Oracle HTTP Server has a different mechanism for the basic authentication mode. The user name and password must be stored in the DAD. Oracle HTTP Server uses HTTP basic authentication where the credentials are stored in a password file on the file system. Authentication is verified against the users listed in that file.

Basic Authentication Mode

mod_plsql supports basic authentication. Oracle HTTP Server authenticates users' credentials against a password file on the file system. This functionality is provided by a module called `mod_auth`.

3.1.3 Global OWA, Custom OWA, and Per Package (Custom Authentication)

Custom authentication enables applications to authenticate users within the application itself, not at the database level. Authorization is performed by invoking a user-written authorization function.

Custom authentication is done using OWA_CUSTOM and cannot be combined with dynamic username/password authentication. Custom authentication needs to have a static username/password stored in the DAD configuration file. mod_plsql uses this DAD username/password to login to the database. Once mod_plsql is logged in, authentication control is passed back to the application, by calling an application-level PL/SQL hook. This callback function is implemented by the application developers. The value returned by the callback function determines the success or failure of authentication. The value `TRUE` means success and `FALSE` means failure.

Depending on the kind of custom authentication needed, you can place the authentication function in different locations:

- Global OWA enables you to invoke the same authentication function for all users and procedures.
- Custom OWA enables you to invoke a different authentication function for each user and for all procedures.
- Per Package authentication enables you to invoke the authentication function for all users, but only for procedures in a specific package or for anonymous procedures.

For example, when using Custom OWA, an authorization function might verify that a user has logged in as user **guest** with password **welcome**, or it might check the user's IP address to determine access.

Table 3–2 summarizes the parameter values.

Table 3–2 Custom Authentication Modes and Callback Functions

Mode	Access Control Scope	Callback Function
Global OWA	All packages	`owa_custom.authorize` in the OWA package schema.
Custom OWA	All packages	`owa_custom.authorize` in the user's schema, or, if not found, in the OWA package schema.
Per package	Specified package	`packageName.authorize` in the user's schema, or `anonymous.authorize` is called.

3.2 Deauthenticating Users

For DADs using dynamic authentication (no username/password in the DAD), mod_plsql allows users to log off (clear HTTP authentication information) programmatically through a PL/SQL procedure without having to exit all browser instances. This feature is supported on Netscape 3.0 or higher and on Microsoft Internet Explorer. On other browsers, the user may have to exit the browser to deauthenticate.

Deauthentication can be done programatically by creating your own logout procedure, which simulates a logout and redirects the user to a sign-off page.

Create or replace procedure *MyLogOffProc* as follows:

```
BEGIN
   -- Open the HTTP header
   owa_util.mime_header('text/html', FALSE, NULL);

   -- Send a cookie to logout
   owa_cookie.send('WDB_GATEWAY_LOGOUT', 'YES', path=>'/');

   -- Close the HTTP header
   owa_util.http_header_close;

   -- Generate the page
   htp.p('You have been logged off from the WEBSITE');
   htp.anchor( 'http://www.abc.com', 'click here');
   htp.p('<BR>bye');
END;
```

Another method of deauthentication is to add `/logmeoff` after the DAD in the URL. For example:

```
http://www.abc.com:2000/pls/myDAD/logmeoff
```

3.3 Protecting the PL/SQL Procedures Granted to PUBLIC

Every database package granted to public can be directly executed using the following URL:

```
http://hostname:port/pls/dad/schema.package.procedure
```

With the different levels of authentication, you must protect the execution of the PL/SQL procedures granted to PUBLIC in the database. These procedures (in the dbms_% packages, utl_% packages, and all packages under the SYS schema) pose a security vulnerability when they are executed through a Web browser. Such packages are intended only for the PL/SQL application developer.

3.3.1 Using the PlsqlExclusionList Directive in mod_plsql

mod_plsql provides a DAD parameter directive called *PlsqlExclusionList* to protect the execution of these PL/SQL packages and other packages that are specific to applications. The *PlsqlExclusionList* directive specifies a pattern for procedure, package, and schema names that are forbidden from being directly executed from a browser. This is a multiline directive in which each pattern is specified on one line. The pattern is not case sensitive and it accepts simple wildcards such as `*`, `?`, and `[a-z]`. The default patterns that are not accessible from a direct URL are `sys.*`, `dbms_*`, `utl_*`, and `owa_util.*`.

> **Caution:** Setting the *PlsqlExclusionList* directive to **#NONE#** will disable all protection. It is not recommended for an active Web site. Only use this setting for debugging purposes.

If the *PlsqlExclusionList* directive is overridden, the default settings do not apply. In this case, you must add the default list to the list of excluded patterns.

3.3.2 Accessing the PlsqlExclusionList Directive

You can set the *PlsqlExclusionList* directive in the mod_plsql configuration file called `dads.conf`. This configuration file is located in the following directories:

- (UNIX) *ORACLE_HOME*`/Apache/modplsql/conf/`
- (Windows) *ORACLE_HOME*`\Apache\modplsql\conf`

Where *ORACLE_HOME* is the location of your Oracle HTTP Server installation.

To ensure the best security for PL/SQL procedures that are granted to PUBLIC, specify the system default settings with the *PlsqlExclusionList* directive in the `dads.conf` file as shown in Example 3–1.

Example 3–1 System Default Settings Specified with the PlsqlExclusionList Directive

```
PlsqlExclusionList sys.*
PlsqlExclusionList dbms_*
PlsqlExclusionList utl_*
PlsqlExclusionList owa_util.*
PlsqlExclusionList owa.*
PlsqlExclusionList htp.*
PlsqlExclusionList htf.*
PlsqlExclusionList myschema.mypackage*
```

In addition to the patterns that are specified with this directive, mod_plsql also disallows any URLs that contain special characters such as tabs, newlines, carriage returns, single quotation marks ('), or reverse slashes (\). You cannot change this.

4

Optimizing PL/SQL Performance

This chapter discusses the techniques for improving PL/SQL performance in Oracle HTTP Server.

This chapter contains the following sections:

- PL/SQL Performance in Oracle HTTP Server - Overview
- Process-Based and Thread-Based Operation in Oracle HTTP Server
- Performance Tuning Issues for mod_plsql
- Performance Tuning Areas in mod_plsql
- Using Caching with PL/SQL-Based Web Applications
- Tuning File System Cache to Improve Caching Performance
- Oracle HTTP Server Directives

4.1 PL/SQL Performance in Oracle HTTP Server - Overview

This chapter describes several techniques to improve the performance of PL/SQL based Web applications in Oracle HTTP Server.

Table 4–1 lists recommendations for Database Access Descriptor (DAD) parameters and settings. By default, these DAD parameters are specified in the file `dads.conf`. On UNIX systems, this is in the *ORACLE_HOME*`/Apache/modplsql/conf` directory. On Windows systems, by default, this file is in the directory *ORACLE_HOME*`\Apache\Apache\modplsql\conf` directory. The file `dads.README` in this directory describes the DAD parameters in detail.

Table 4–1 Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlAlwaysDescribeProcedure`	Set this to `off` for best performance. Default Value: `off`
`PlsqlDatabaseConnectString`	For newer DADs, use the `ServiceNameFormat`. Use the `SIDFormat` only for backward compatibility. Note: for HA configurations of the database, it is recommended that the connect string parameter gets resolved through an LDAP lookup.
`PlsqlFetchBufferSize`	For multibyte character sets like Japanese or Chinese, setting this to `256` should provide better performance Default Value: `128`

Table 4–1 (Cont.) Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlIdleSessionCleanupInterval`	Increasing this parameter allows pooled database connections to remain available, in the pool, for the specified time Default Value: `15` (minutes) **See Also:** Section 4.3.3, "Restarting the mod_plsql Connection Pool"
`PlsqlLogEnable`	This parameter should be set to `Off` unless recommended by Oracle Support Services for debugging purposes Default Value: `off`
`PlsqlMaxRequestsPerSession`	If the PL/SQL based Web application does not leak resources or memory, this parameter can be set to a higher value (for example, 5000). Default Value: `1000` **See Also:** Section 4.3.2, "Closing Pooled Database Sessions" and Section 4.4.2, "Connection Pooling Tips and Oracle HTTP Server Configuration".
`PlsqlNLSLanguage`	Set this parameter to match the database Globalization Support parameters to eliminate overheads in character set conversions in Oracle Net Services
`PlsqlSessionStateManagement`	Set this parameter to `StatelessWithFastResetPackageState` if the database is 8.1.7.2 or later.

Table 4–2 lists mod_plsql caching options and the sections that describe these caching options.

Table 4–2 Caching Options

Option	Description
Expires Technique	Best performance - for content that changes predictably. **See Also:** Section 4.5.2, "Using the Expires Technique"
Validation technique	Good performance - for content that changes unpredictably. **See Also:** Section 4.5.1, "Using the Validation Technique"
System-level caching	Improves performance by caching one copy for everyone on system. **See Also:** Section 4.5.3, "System- and User-Level Caching with PL/SQL-Based Web Applications"

See Also: Chapter 6 "Oracle HTTP Server Modules" in the *Oracle HTTP Server Administrator's Guide* for details on the DAD parameters shown in Table 4–1.

4.2 Process-Based and Thread-Based Operation in Oracle HTTP Server

This chapter describes PL/SQL performance issues that apply on platforms where the Oracle HTTP Server is process-based and thread-based. On a process-based Oracle HTTP Server, such as those running on UNIX-based platforms, each process servers all types of HTTP requests, including servlets and PL/SQL, static files. On a thread-based Oracle HTTP Server, such as Windows-based platforms, there is just one Oracle HTTP

Server process with multiple threads within the process; individual threads can be used to serve all types of HTTP requests.

Note: In some cases in this chapter, we make references to performance optimizations that apply to PL/SQL-based Web applications where the distinction between platforms, either process-based or thread based, is significant.

4.3 Performance Tuning Issues for mod_plsql

When tuning mod_plsql to improve the performance of PL/SQL based Web applications, it is important to be familiar with some mod_plsql internals. This section presents a basic overview of some mod_plsql functionality.

This section covers the following topics:

- Connection Pooling with mod_plsql
- Closing Pooled Database Sessions
- Restarting the mod_plsql Connection Pool

4.3.1 Connection Pooling with mod_plsql

The Database Server connection pooling logic supplied with mod_plsql can be best explained with an example.

For example, consider the following typical scenario:

1. The Oracle HTTP Server listener is started. There are no database connections in the connection pool maintained by mod_plsql.
2. A browser makes a mod_plsql request (R1) for Database Access Descriptor (DAD) D1.
3. One of the Oracle HTTP Server processes (`httpd` process P1) starts servicing the request R1.
4. mod_plsql in process P1 checks its connection pool and finds that there are no database connections in its pool for that user request.
5. Based on the information in DAD D1, mod_plsql in process P1 opens a new database connection, services the PL/SQL request, and adds the database connection to its pool.
6. From this point on, all subsequent requests to process P1 for DAD D1 can now make use of the database connection pooled by mod_plsql.
7. If a request for DAD D1 gets picked up by another process (process P2), then mod_plsql in process P2 opens its own database connection, services the request, and adds the database connection to its pool.
8. From this point on, all subsequent requests to process P2 for DAD D1 can now make use of the database connection pooled by mod_plsql.
9. Now, assume that a request R2 is made for DAD D2 and this request gets routed to process P1.
10. mod_plsql in process P1 does not have any database connections pooled for DAD D2, and a new database session is created for DAD D2 and pooled after servicing the request. Process P1 now has two database connections pooled, one for DAD D1 and another for DAD D2.

The important details in the example shown in steps 1-10 are:

a. Each Oracle HTTP Server process serves all types of requests, such as static files requests, servlet requests, and mod_plsql requests. There is no control on which Oracle HTTP Server process services the next request.

b. One Oracle HTTP Server process cannot use or share the connection pool created by another process.

> **Note:** On Windows systems, the Oracle HTTP Server is just one process. Therefore, the connection pool is shared and can be used across threads.

c. Each Oracle HTTP Server process pools at most one database connection for each DAD.

d. User sessions are switched within a pooled database connection for a DAD. For DADs based on Oracle Application Server Single Sign-On (SSO), proxy authentication is used to switch the user session. For non-SSO users, using HTTP basic authentication with the username and password not in the DAD, users are re-authenticated on the same connection.

e. Multiple DADs may point to the same database instance, but database connections are not shared across DADs even within the same process.

f. Unused DADs do not result in any database connections.

In the worst-case scenario, the total number of database connections pooled by mod_plsql is a factor of the total number of active DADs multiplied by the number of Oracle HTTP Server (`httpd`) processes running at any given time for a single Oracle HTTP Server instance. If you have configured the Oracle HTTP Server processes to a high number, you need to configure the back-end database to handle a corresponding number of database sessions, and remember that this configuration value needs to be multiplied times the number of Oracle HTTP Server instances that use the back-end database.

For example, if there are three Oracle HTTP Server instances configured to spawn a maximum of 50 `httpd` processes each, plus two active DADs, you need to set up the database to allow 300 (`3*50*2`) sessions. This number does not include any sessions that are needed to allow other Web applications to connect.

On UNIX systems, database connections cannot be shared across `httpd` processes, and process-based platforms have more of a *Connection Reuse* feature than *Connection Pooling*. Note that this is an artifact of the process-model in Oracle HTTP Server.

> **Note:** Refer to Section 4.4.4, "Two-Listener Strategy" if the number of database sessions is a concern for details on how to address this problem.
>
> On Windows systems, the Oracle HTTP Server runs as a single process, and the connection pool is shared. Therefore, Section 4.4.4, "Two-Listener Strategy" does not apply to Windows systems.

4.3.2 Closing Pooled Database Sessions

Pooled database sessions are closed under the following circumstances:

1. When a pooled connection has been used to serve a configured number of requests.

   By default each connection pooled by mod_plsql is used to service a maximum of 1000 requests and then the database connection is shut down and reestablished. This is done to make sure that any resource leaks in the PL/SQL based Web application, or in the Oracle client server side, do not adversely affect the system. To change the default value of `1000` by tuning the DAD configuration parameter `PlsqlMaxRequestsPerSession`.

2. When a pooled connection has been idle for an extended period of time.

   By default, each pooled connection gets automatically cleaned up after 15 minutes of idle time. This operation is performed by the cleanup thread in mod_plsql. For heavily loaded sites, each connection could be used at least once every 15 minutes and the connection cleanup might not happen for a long period of time. In such a case, the connection would be cleaned up based on the configuration value of `PlsqlMaxRequestsPerSession`. Change the default value of 15 minutes by tuning the mod_plsql configuration parameter `PlsqlIdleSessionCleanupInterval`. Consider increasing the default for better performance in cases where the site is not heavily loaded.

3. On UNIX systems, when the Oracle HTTP Server process goes down.

   On UNIX systems, the Oracle HTTP Server configuration parameter `MaxRequestsPerChild` governs when an Oracle HTTP Server process will be shut down. For example, if this parameter is set to `5000`, each Oracle HTTP Server process would serve exactly 5000 requests before it is shut down. Oracle HTTP Server processes could also start up and shut down as part of Oracle HTTP Server maintenance based on the configuration parameters `MinSpareServers`, `MaxSpareServers`, and `MaxClients`. For mod_plsql connection pooling to be effective, it is extremely important that Oracle HTTP Server is configured so that each Oracle HTTP Server process remains active for some period of time. An incorrect configuration of Oracle HTTP Server could result in a setup where Oracle HTTP Server processes are heavily started up and shut down. Such a configuration would require that each new Oracle HTTP Server process replenish the connection pool before subsequent requests gain any benefit of pooling.

   See Also: Chapter 6 "Oracle HTTP Server Modules" in the *Oracle HTTP Server Administrator's Guide.*

4.3.3 Restarting the mod_plsql Connection Pool

This depends primarily on the amount of time the database is shut down. If the database is restarted after more than 15 minutes from being shut down, the users do not experience any problems when trying to use the Oracle HTTP Server listener. This is because the cleanup thread in mod_plsql cleans up database sessions that are unused for more than 15 minutes. The time specified for cleaning up idle sessions is tunable using the `PlsqlIdleSessionCleanupInterval`, configuration parameter (the default value is 15 minutes).

If the database is restarted in less than 15 minutes, then a few initial requests return with errors, but the system quickly becomes usable again. The number of requests that experience failure is equal to the number of connections that were pooled by mod_plsql.

See Also: Table 4–1, " Database Access Descriptor (DAD) Parameters Recommended Setting Summary"

4.4 Performance Tuning Areas in mod_plsql

While using mod_plsql, there are three areas that affect performance and scalability:

- PL/SQL Based Web Application Development Considerations
- Connection Pooling Tips and Oracle HTTP Server Configuration
- Tuning the Number of Database Sessions

4.4.1 PL/SQL Based Web Application Development Considerations

PL/SQL Gateway users should consider the following topics when developing PL/SQL based Web applications:

1. **Manage the use of Database Access Descriptors (DADs)**

   Try to restrict the number of DADs that each Oracle HTTP Server node uses.

   Note: Performance is not affected if there are DADs that are not being used.

2. **Use of Nested Tables**

   PL/SQL provides the ability to create tables. To build PL/SQL tables, you build a table that gives the datatype of the table, as well as the index of the table. The index of the table is the binary integer ranging from -2147483647 to +2147483647. This table index option is known as *sparsity*, and allows meaningful index numbers such as customer numbers, employee number, or other useful index keys. Use PL/SQL tables to process large amounts of data.

   PL/SQL provides `TABLE` and `VARRAY` (variable size array) collection types. The `TABLE` collection type is called a nested table. Nested tables are unlimited in size and can be sparse, which means that elements within the nested table can be deleted using the `DELETE` procedure. Variable size arrays have a maximum size and maintain their order and subscript when stored in the database. Nested table data is stored in a system table that is associated with the nested table. Variable size arrays are suited for batch operations in which the application processes the data in batch array style. Nested tables make for efficient queries by storing the nested table in a storage table, where each element maps to a row in the storage table.

3. **Use procedure naming overloading with caution**

   PL/SQL based Web applications should use the procedure name overloading feature with caution. It is best if procedure name overloading is avoided by having multiple procedures with different names.

4. **Consider rewriting applications where there is significant overhead in determining the type parameters**

   PL/SQL based Web applications should be aware of the overhead in trying to execute procedures where the URL does not provide enough details to know about the type of the parameter, such as scalar or array. In such cases, the first attempt to execute a procedure fails and the procedure signature needs to be described before it can be executed by mod_plsql.

5. **Use procedures with 2-parameter style flexible parameter passing**

   Procedures should make use of the more performant 2-parameter style flexible parameter passing rather than the 4-parameter style parameter passing

See Also: *Oracle Database Application Developer's Guide - Fundamentals*

4.4.2 Connection Pooling Tips and Oracle HTTP Server Configuration

Consider the following topics when configuring connection pooling with Oracle HTTP Server:

1. Using the default connections pooling and setting values for `PlsqlMaxRequestsPerSession`

   Creating new database connections is an expensive operation and it is best if every request does not have to open and close it own database connections. The optimal technique is to make sure that database connections opened in one request are reused in subsequent requests. In some rare situations, where a database is accessed very infrequently and performance is not a major concern, connection pooling can be disabled. For example, if the administrator accesses a site infrequently to perform some administration tasks, then the DAD used to access the administrator applications can choose to disable connection pooling. To disable connection pooling, set the DAD parameter `PlsqlMaxRequestsPerSession` to the value 1.

   Note: Setting `PlsqlMaxRequestsPerSession` to the value 1 reduces the number of available database sessions and may impact performance.

2. On UNIX systems, Oracle HTTP Server configuration should be properly tuned so that once processes are started up, the processes remain up for a while. Otherwise, the connection pooling in mod_plsql is rendered useless. The Oracle HTTP Server listener should not have to continually start up and shut down processes. A proper load analysis should be performed of the site to determine what the average load on the Web site. The Oracle HTTP Server configuration should be tuned such that the number of `httpd` processes can handle the average load on the system. In addition, the configuration parameter `MaxClients` in the `httpd.conf` file should be able to handle random load spikes as well.

3. On UNIX systems, Oracle HTTP Server processes should be configured so that processes are eventually killed and restarted. This is required to manage any possible memory leaks in various components accessed through the Oracle HTTP Server. This is specifically required in mod_plsql to ensure that any database session resource leaks do not cause a problem. Make sure that `MaxRequestsPerChild` configuration parameter is set to a high number. For PL/SQL based Web applications, this should not be set to `0`.

   Note: On Windows systems, to assure that database session resource leaks do not cause problems, the Oracle HTTP Server needs to be periodically restarted.

4. For heavily loaded sites, the Oracle HTTP Server configuration parameter `KeepAlive` should be disabled. This ensures that each process is available to service requests from other clients as soon as a process is done with servicing the current request. For sites which are not heavily loaded, and where it is guaranteed that the number of Oracle HTTP Server processes are always greater than the number of simultaneous requests to the Oracle HTTP Server listener, enabling the

KeepAlive parameter results in performance improvements. In such cases, make sure to tune the KeepAliveTimeout parameter appropriately.

5. You may want to lower the value of Timeout in the Oracle HTTP Server configuration. This ensures that Oracle HTTP Server processes are freed up earlier if a client is not responding in a timely manner. Do not set this value too low, otherwise slower responding clients could time out.

6. Most Web sites have many static image files, which are displayed in each screen for a consistent user interface. Such files rarely change and you can reduce a considerable load on the system by tagging each image served by the Oracle HTTP Server listener with mod_expires. You should also consider front-ending your Web site with Oracle Application Server Web Cache.

   - How do I know if the Web site can benefit from the use of mod_expires?
     - Use Netscape, or any browser that enables you to view page caching information, and visit several heavily accessed Web pages on the site. On each page, right click the mouse and select View Info from the pop up menu (or the equivalent command for your browser). If the top panel in the page information window lists many different images and static content, then the site could benefit from the use of mod_expires.
     - You can also check the Oracle HTTP Server access logs to see what percentage of requests result in HTTP 304 (Not Modified) status. Use the grep utility to search for 304 in the access_log and divide this resulting number of lines by the total number of lines in the access_log. If this percentage is high, then the site could benefit from the use of mod_expires.
   - How do I tag static files with the Expires header?
     - Locate the Location directive used to serve your static image files. Add the ExpiresActive and ExpiresDefault directives to it.

```
Alias /images/ "/u01/app/oracle/myimages/"
<Directory "/u01/app/oracle/myimages/">
     AllowOverride None
     Order allow, deny
     Allow from all
     ExpiresActive On
     ExpiresDefault A2592000
</Directory>
```

     - The browser caches all static files served off the /images path for 30 days from now. Refer to the *Oracle HTTP Server Administrator's Guide* for more details.

   - How do I know if the static files are being tagged with the Expires header?
     - Using Netscape, or the browser of your choice, clean up all the cached files in the browser.
     - Visit a Web page that should have images tagged with the Expires header. Right click the mouse on the page and select **View Info**, from the pop up menu. or use the equivalent command for your browser.
     - In the top panel of the page information, select an image that should be tagged with the Expires header.

– Review the information displayed in the bottom panel. The `Expires` header should be set to a valid date. If this entry is `No date given`, then the file is not being tagged with the `Expires` header.

4.4.3 Tuning the Number of Database Sessions

Consider the following topics when tuning the number of database sessions:

1. The `processes` and `sessions` parameters in the Oracle `init$SID.ora` configuration file should be set so that Oracle is able to handle the maximum number of database sessions. This number should be proportional to the number of DADs times the maximum number of Oracle HTTP Server processes, times the number of Oracle HTTP Server instances.
2. Using a two-listener strategy or using a shared server reduces the number of database sessions. See Section 4.4.4, "Two-Listener Strategy".
3. On UNIX platforms, the connection pool is not shared across Oracle HTTP Server processes. For this reason, it is recommended that the application use as few DADs as possible.

4.4.4 Two-Listener Strategy

On platforms where the Oracle HTTP Server is process-based, such as all UNIX-based platforms, each process serves all types of HTTP requests, including servlets, PLSQL, static files, and CGI. In a single Oracle HTTP Server listener setup, each `httpd` process maintains its own connection pool to the database. The maximum number of database sessions is governed by the setting in `httpd.conf` configuration file for `StartServers`, `MinSpareServers`, and `MaxSpareServers`, plus the load on the system. This architecture does not allow for tuning the number of database sessions based on the number of mod_plsql requests. To tune the number of database sessions based on the number of mod_plsql requests, install a separate HTTP listener for mod_plsql requests only. This approach greatly reduces the number of database sessions that are needed to serve mod_plsql requests.

For example, assume a main Oracle HTTP Server listener is running on port 7777 of `mylsnr1.mycompany.com`. First, you can install another Oracle HTTP Server listener on port 8888 on `mylsnr2.mycompany.com`. Next, redirect all mod_plsql requests made to `mylsnr1.mycompany.com:7777` to the second listener on `mylsnr2.mycompany.com:8888`. Review the following steps:

1. To redirect all PL/SQL requests for `mylsnr1.mycompany.com:7777` to `mylsnr2.mycompany.com:8888`, make the following configuration changes:
   a. For the Oracle HTTP Server listener running on Port 7777, edit *`ORACLE_HOME`*`/Apache/modplsql/conf/plsql.conf` file. Comment out the following line by putting a `#` in front of the line:

      ```
      #LoadModule plsql_module...
      ```

   b. Copy the DAD location used to service PL/SQL requests in `mylsnr1.mycompany.com` to the configuration file *`ORACLE_HOME`*`/Apache/modplsql/conf/dads.conf` in `mylsnr2.mycompany.com`.

      Comment out the DAD location configuration parameters on `mylsnr1.mycompany.com` by prefixing the line with a "#" character.

      ```
      #<Location /pls/dad>
      #...
      ```

```
#</Location>
```

c. Configure this listener to forward all mod_plsql requests for this DAD location to the second listener by adding the following line in `dads.conf`:

```
ProxyPass /pls/dad http://mylsnr2.mycompany.com:8888/pls/dad
```

Repeat the configuration procedures for all DAD Locations.

2. Because the PL/SQL procedures generate URLs that are displayed in the browser, it is important that all URLs are constructed without any references to the internal mod_plsql listener on `mylsnr2.mycompany.com:8888`. Depending on how the URLs are being generated in the PL/SQL based Web application, there are three options:

   - If the URLs are hard-coded into the application, make sure that they are always generated using the hard-coded values as HOST=mylsnr1.mycompany.com and PORT=7777. No change would be required for this scenario.
   - If the PL/SQL based Web applications always use the CGI environment variables `SERVER_NAME` and `SERVER_PORT`, then it is easy to change the configuration of the listener on `mylsnr2.mycompany.com`. Edit the file and change the lines `ServerName` and `Port` in the *`ORACLE_HOME`*`/Apache/Apache/conf/httpd.conf` file for the second listener as follows:

     ```
     ServerName mylsnr1.mycompany.com (was mylsnr2.mycompany.com)
     Port 7777                        (was 8888)
     ```

   - If the URLs are being generated using the CGI environment variable `HTTP_HOST`, you need to override the CGI environment variables for the Oracle HTTP Server listener running on Port 8888. Add the following lines to the *`ORACLE_HOME`*`/Apache/modplsql/conf/dads.conf` file for each DAD to override the default CGI environment variables `HOST`, `SERVER_NAME`, and `SERVER_PORT`:

     ```
     PlsqlCGIEnvironmentList  SERVER_NAME  mylsnr1.mycompany.com
     PlsqlCGIEnvironmentList  SERVER_PORT  7777
     PlsqlCGIEnvironmentList  HOST mylsnr1.us.oracle.com:7777
     ```

   In all cases, the intent is to fool the application to generate URLs as if there never was a second listener.

3. Test the setup and make sure that you can access all the DADs without any problems.

4. In this setup, the main listener `mylsnr1.mycompany.com` can be configured based on the total load on the Oracle HTTP Server listener. The second listener on `mylsnr2.mycompany.com` can be fine-tuned based on just the mod_plsql requests being made.

4.4.5 Overhead Problems

While executing some of the stored procedures, mod_plsql may incur a `Describe` overhead, which would result in two extra round trips to the database for a successful execution. This has performance implications.

4.4.5.1 The Describe Overhead

In order to execute stored procedures, mod_plsql needs to know about the datatype of the parameters being passed in. Based on this information, mod_plsql binds each parameter either as an array or as a scalar. One way to know the procedure signature is to describe the procedure before executing it. However, this approach is not efficient because every procedure has to be described before execution. To avoid the describe overhead, mod_plsql looks at the number of parameters passed for each parameter name. It uses this information to assume the datatype of each variable. The logic is simply that if there is a single value being passed, then the parameter is a scalar, otherwise it is an array. This works for most cases but fails if there is an attempt to pass a single value for an array parameter or pass multiple values for a scalar. In such cases, the first attempt to execute the PL/SQL procedure fails. mod_plsql issues a `Describe` call to get the signature of the PL/SQL procedure and binds each parameter based on the information retrieved from the `Describe` operation. The procedure is re-executed and results are sent back.

This `Describe` call occurs transparently to the procedure, but internally mod_plsql has encountered two extra round trips, one for the failed execute call and the other for the describe call.

4.4.5.2 Avoiding the Describe Overhead

You can avoid performance problems with the following:

- Use flexible parameter passing.
- Always ensure that you pass multiple values for arrays. For single values, you can pass dummy values that are ignored by the procedure.
- Use the following workaround, which defines a two-parameter style procedure which defaults the unused variables.

  1. Define a scalar equivalent of your procedure, which internally calls the original procedure. For example, the original package could be similar to the following example:

     ```
     CREATE OR REPLACE PACKAGE testpkg AS
      TYPE myArrayType is TABLE of VARCHAR2(32767) INDEX BY binary_ integer;
      PROCEDURE arrayproc (arr myArrayType);
     END testpkg;
     /
     ```

  2. If you are making URL calls like `/pls/.../testpkg.arrayproc? arr= 1`, change the specification to be similar to the following:

     ```
     CREATE OR REPLACE PACKAGE testpkg AS
      TYPE myArrayType is TABLE of VARCHAR2( 32767) INDEX BY binary_integer;
        PROCEDURE arrayproc (arr varchar2);
        PROCEDURE arrayproc (arr myArrayType);
     END testpkg;
     /
     ```

  3. The procedure `arrayproc` should be similar to:

     ```
     CREATE OR REPLACE PACKAGE BODY testpkg AS
     PROCEDURE arrayproc (arr varchar2) IS
       localArr myArrayType;
     BEGIN
       localArr( 1) := arr;
       arrayproc (localArr);
     END arrayproc;
     ```

4.4.6 The Flexible Parameter Passing (four-parameter) Overhead

Round-trip overhead exists if a PL/SQL procedure is using the older style four-parameter interface. The PL/SQL Gateway first tries to execute the procedure by using the two-parameter interface. If this fails, the PL/SQL Gateway tries the four-parameter interface. This implies that all four-parameter interface procedures experience one extra round-trip for execution.

- Avoiding the flexible parameter passing overhead

  To avoid this overhead, it is recommended that you write corresponding wrappers that use the two-parameter interface and internally call the four-parameter interface procedures. Another option is to change the specification of the original procedure to default to the parameters that are not passed in the two-parameter interface. The four-parameter interface has been provided only for backward compatibility and will be deprecated in the future.

- Using flexible parameters and the exclamation mark

  The flexible parameter passing mode in Oracle HTTP Server expects the PL/SQL procedure to have the exclamation mark before the procedure name. Due to performance implications of the auto-detect method used in Oracle HTTP Server, the exclamation mark is now required for flexible parameter passing in Oracle HTTP Server. In Oracle HTTP Server, each procedure is described completely before being executed. The procedure `Describe` call determines the signature of the procedure and requires around-trip to the database. The PL/SQL Gateway in Oracle HTTP Server avoids this round trip by having end-users explicitly indicate the flexible parameter passing convention by adding the exclamation mark before the procedure.

4.5 Using Caching with PL/SQL-Based Web Applications

Caching can improve the performance of PL/SQL based Web applications. To improve performance, you can cache Web content generated by PL/SQL procedures in the middle-tier and decrease the database workload.

This section covers the techniques used in caching, including the following:

- Using the Validation Technique - An application asks the server if the page has been modified since it was last presented.
- Using the Expires Technique - Based upon a specific time period, the PL/SQL based Web application determines the page will be cached, or should be generated again.
- System- and User-Level Caching with PL/SQL-Based Web Applications - This is valid whether you are using the Validation Technique or the Expires Technique. The level of caching is determined by whether a page is cached for a particular user or for every user in the system.

These techniques and levels are implemented using `owa_cache` packages located inside the PL/SQL Web Toolkit.

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals*

4.5.1 Using the Validation Technique

In general, the validation technique basically asks the server if the page has been modified since it was last presented. If it has not been modified, the cached page will be presented to the user. If the page has been modified, a new copy will be retrieved, presented to the user and then cached.

There are two methods which use the Validation Technique: Last-Modified method, and the Entity Tag method. The next two sections show how these techniques are used in the HTTP protocol. Although the PL/SQL Gateway does not use the HTTP protocol, many of the same principles are used.

4.5.1.1 Last-Modified

When a Web page is generated using the HTTP protocol, it contains a **Last-Modified** Response Header. This header indicates the date, relative to the server, of the content that was requested. Browsers save this date information along with the content. When subsequent requests are made for the URL of the Web page, the browser then:

1. Determines if it has a cached version.
2. Extracts the date information.
3. Generates the Request Header **If-Modified-Since**.
4. Sends the request to the server.

Cache-enabled servers look for the **If-Modified-Since** header and compare it to the date of their content. If the two match, an HTTP Response status header such as "HTTP/1.1 304 Not Modified" is generated, and no content is streamed. After receiving this status code, the browser can reuse its cache entry because it has been validated.

If the two do not match, an HTTP Response header such as "HTTP/1.1 200 OK" is generated and the new content is streamed, along with a new **Last-Modified Response** header. Upon receipt of this status code, the browser must replace its cache entry with the new content and new date information.

4.5.1.2 Entity Tag Method

Another validation method provided by the HTTP protocol is the **ETag** (Entity Tag) Response and Request header. The value of this header is a string that is opaque to the browser. Servers generate this string based on their type of application. This is a more generic validation method than the **If-Modified-Since** header, which can only contain a date value.

The **ETag** method works very similar to the Last Modified method. Servers generate the ETag as part of the Response Header. The browser stores this opaque header value along with the content that is steamed back. When the next request for this content arrives, the browser passes the **If-Match** header with the opaque value that it stored to the server. Because the server generated this opaque value, it is able to determine what to send back to the browser. The rest is exactly like the **Last-Modified** validation method as described earlier.

4.5.1.3 Using the Validation Technique for mod_plsql

Using HTTP validation caching as a framework, the following is the Validation Model for mod_plsql.

PL/SQL based Web applications that want to control the content being served should use this type of caching. This technique offers some moderate performance gains. One example of this would be a Web application that serves dynamic content that could

change at any given time. In this case, the Web application needs full control over what is being served. Validation caching always asks the Web application whether the cached content is stale or not before serving it back to the browser.

Figure 4–1 shows the use of the validation technique for mod_plsql.

1. The Oracle HTTP Server receives a PL/SQL procedure request from a client server. The Oracle HTTP Server routes the request to mod_plsql.
2. mod_plsql prepares the request.
3. mod_plsql invokes the PL/SQL procedure in the Web application. mod_plsql passes the usual Common Gateway Interface (CGI) environment variables to the Web application.
4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the `owa_cache` procedure from the PL/SQL Web Toolkit to set the tag and cache level:

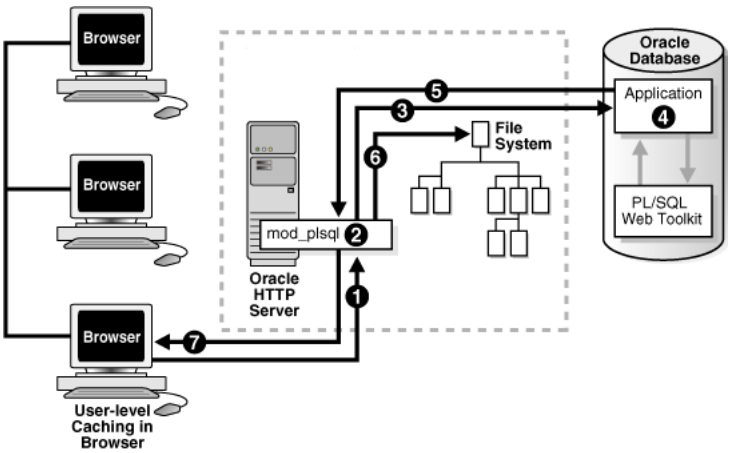
```
owa_cache.set_cache(p_etag, p_level);
```

Table 4–3 Validation Model Parameters

Parameter	Description
`set_cache` procedure	Sets up the headers to notify mod_plsql that the content being streamed back can be cached. Then, the mod_plsql caches the content on the local file system along with the tag and caching level information as it is streamed back to the browser.
`p_etag`	The string that the procedure generates to tag the content.
`p_level`	The caching level: `SYSTEM` for system level or `USER` for user level.

1. The HTML is returned to mod_plsql.
2. mod_plsql stores the cacheable content in its file system for the next request.
3. The Oracle HTTP Server sends the response to the client browser.

Figure 4–1 Validation Technique



4.5.1.4 Second Request Using the Validation Technique

Using the Validation Technique for mod_plsql, a second request is made by the client browser for the same PL/SQL procedure.

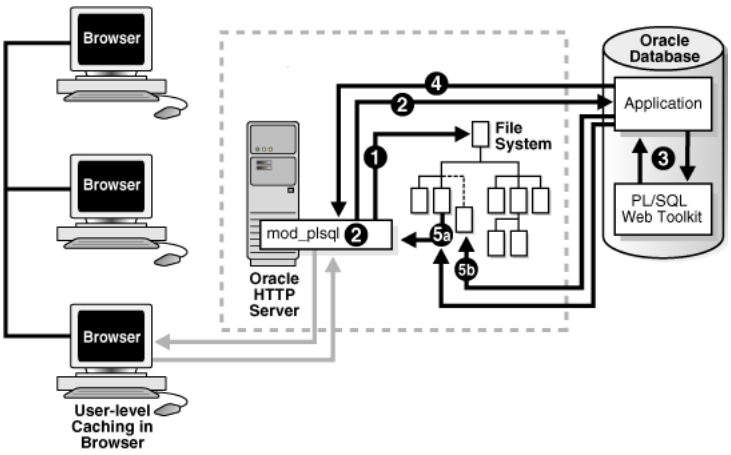
Figure 4–2 shows the second request using the Validation Technique.

1. mod_plsql detects that it has a cached content for the request.
2. mod_plsql forwards the same tag and caching level information (from the first request) to the PL/SQL procedure as part of the CGI environment variables.
3. The PL/SQL procedure uses these caching CGI environment variables to check if the content has changed. It does so by calling the following `owa_cache` functions from the PL/SQL Web Toolkit:

   ```
   owa_cache.get_etag;
   owa_cache.get_level;
   ```

   These owa functions get the tag and caching level.
4. The Web application sends the caching information to mod_plsql.
5. Based on that information determines whether the content needs to be regenerated or can be served from the cache.
   a. If the content is still the same, the procedure calls the `owa_cache.set_not_modified` procedure and generates no content. This causes mod_plsql to use its cached content. The cached content is directly streamed back to the browser.
   b. If the content has changed, it generates the new content along with a new tag and caching level. mod_plsql replaces its stale cached copy with a new one and updates the tag and caching level information. The newly generated content is streamed back to the browser.

Figure 4–2 Validation Technique-Second Request



4.5.2 Using the Expires Technique

In the validation model, mod_plsql always asks the PL/SQL procedure if it can serve the content from the cache. In the expires model, the procedure pre-establishes the content validity period. Therefore, mod_plsql can serve the content from its cache without asking the procedure. This further improves performance because no interaction with the database is required.

This caching technique offers the best performance. Use if your PL/SQL based Web application is not sensitive to serving stale content. One example of this is an application that generates news daily. The news can be set to be valid for 24 hours. Within the 24 hours, the cached content is served back without contacting the

application. This is essentially the same as serving a file. After 24 hours, mod_plsql will again fetch new content from the application.

Assume the same scenario described for the Validation model, except the procedure uses the Expires model for caching.

Figure 4–3 shows the use of the expires technique for mod_plsql.

1. The Oracle HTTP Server receives a PL/SQL Server Page request from a client server. The Oracle HTTP Server routes the request to mod_plsql.
2. The request is forwarded by mod_plsql to the Oracle Database.
3. mod_plsql invokes the PL/SQL procedure in the application and passes the usual Common Gateway Interface (CGI) environment variables to the application.
4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the `owa_cache` procedure from the PL/SQL Web Toolkit to set the validity period and cache level:

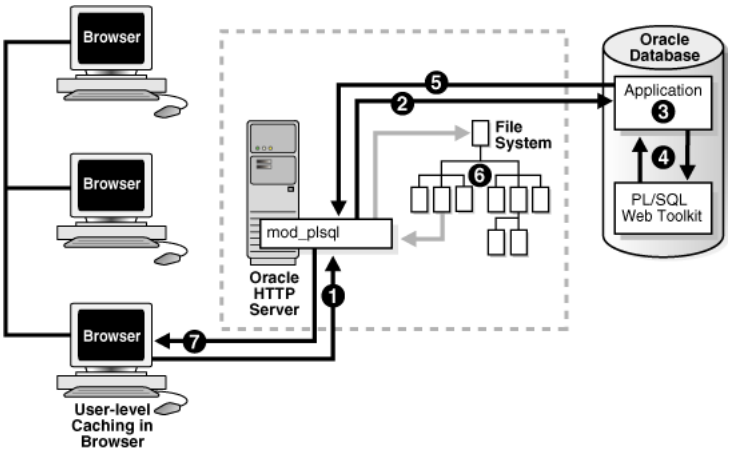
```
owa_cache.set_expires(p_expires, p_level);
```

Table 4–4 Expires Model Parameters

Parameter	Description
`set_expires` procedure	Sets up the headers to notify mod_plsql that Expires caching is being used. mod_plsql then caches the content to the file system along with the validity period and caching level information.
`p_expires`	Number of minutes that the content is valid.
`p_level`	Caching level.

1. The HTML is returned to mod_plsql.
2. mod_plsql stores the cacheable content in its file system for the next request.
3. The Oracle HTTP Server sends the response to the client browser.

Figure 4–3 The Expires Technique



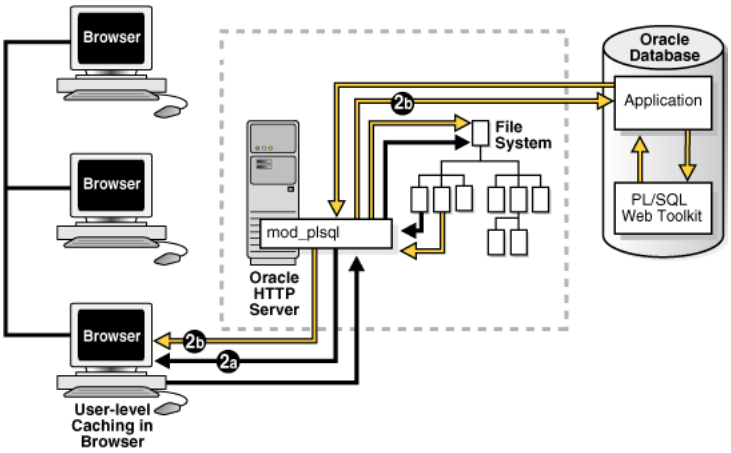
Second Request Using the Expires Technique

Using the same expires model explained earlier, a second request is made by the client browser for the same PL/SQL procedure.

Figure 4–4 shows the second request using the Expires Technique.

1. mod_plsql detects that it has a cached copy of the content that is expires-based.
2. mod_plsql checks the content's validity by taking the difference between the current time and the time this cache file was created.
   a. If this difference is within the validity period, the cached copy is still fresh and will be used without any database interaction. The cached content is directly streamed back to the browser.
   b. If the difference is not within the validity period, the cached copy is stale. mod_plsql invokes the PL/SQL procedure and generates new content. The procedure then decides whether to use expires-based caching again. If so, it also determines the validating period for this new content. The newly generated content is streamed back to the browser.

Figure 4–4 The Expires Technique-Second Request



4.5.3 System- and User-Level Caching with PL/SQL-Based Web Applications

A PL/SQL procedure determines whether generated content is system-level content or user-level. This helps the PL/SQL Gateway cache to store less redundant files if more than one user is looking at the same content. It decides this by:

- For **system-level** content, the procedure passes the string `SYSTEM` as the caching level parameter to the `owa_cache` functions (`set_cache` for validation model or `set_expires` for expires model). This is for every user that shares the cache.

  By using system-level caching, you can save both space in your file system and time for all users in the system. One example of this would be a Web application that generates content that is intended for everybody using the Web application. By caching the content with the system-level setting, only one copy of the content is cached in the file system. Furthermore, every user on that system benefits since the content is served directory from the cache.

- For **user-level** content, it passes the string `USER` as the parameter for the caching level. This is for a specific user that is logged in. The stored cache is unique for that user. Only that user can use the cache. The type of user is determined by the authentication mode. Refer to the following table for the different types of users.

Table 4–5 Type of User Determined by Authentication Mode

Authentication Mode	Type of User
Single Sign-On (SSO)	Lightweight user
Basic	Database user
Custom	Remote user

For example, if no user customizes a PL/SQL based Web application, then the output can be stored in a system-level cache. There will be only one cache copy for every user on the system. User information is not used since the cache can be used by multiple users.

However, if a user customizes the application, a user-level cache is stored for that user only. All other users still use the system level cache. For a user-level cache hit, the user information is a criteria. A user-level cache always overrides a system-level cache.

> **See Also:** Section 3.1, "Authenticating Users Through mod_plsql" for more information on authentication modes.

PL/SQL Web Toolkit Functions (owa_cache package)

Your decision whether to use the Validation technique or the Expires technique determines which `owa_cache` functions to call.

The `owa_cache` package contains procedures to set and get special caching headers and environment variables. These allow developers to use the PL/SQL Gateway cache more easily. This package should already be installed in your database.

Table 4–6 lists the primary functions to call.

Table 4–6 Primary owa_cache Functions

owa Functions	Purpose
`owa_cache.set_cache (p_etag IN varchar2, p_level IN varchar2)`	Validation Model - Sets up the headers. ▪ `p_etag` parameter tags the generated content. ▪ `p_level` parameter is the caching level to use.
`owa_cache.set_not_modified`	Validation Model - Sets up the headers to notify mod_plsql to use the cached content. Only used when a validation -based cache hit occurs.
`owa_cache.get_level`	Validation Model - Gets the caching level, `USER` or `SYSTEM`. Returns null if the cache is not hit.
`owa_cache.get_etag`	Validation Model - Gets the tag associated with the cached content. Returns null if the cache is not hit.
`owa_cache.set_expires( p_expires IN number,` p_level IN varchar2)	Expires Model - Sets up the headers. ▪ `p_expires` parameter is the number of minutes the content is valid. ▪ `p_level` parameter is the caching level to use.

4.6 Tuning File System Cache to Improve Caching Performance

You can configure and use a File System Cache to improve the performance of PL/SQL based Web applications.

This section covers the following topics:

- Introducing File System Cache Tuning
- Enabling File System Cache
- Configuring File System Cache to Reside on a Faster File System
- Resizing File System Cache
- Configuring Cache Cleanup

4.6.1 Introducing File System Cache Tuning

This section covers mod_plsql related File System Cache tuning options. Cache contents are cached using Operating System supplied file system calls; the cached contents are not stored in the mod_plsql memory space. Using the mod_plsql File System Cache, the contents of the cache may be in memory when the Operating System supports, and the system is configured to use features such as memory disk (some UNIX platforms support memory disk based fast storage).

The information in this section can improve the performance of PL/SQL based Web applications when mod_plsql is configured to use the File System Cache.

Table 4–7 lists the cache related parameters that you can set for mod_plsql. Set these parameters in the `cache.conf` file that is available on UNIX in the directory, *ORACLE_HOME*`/Apache/modplsql/conf`, and on Windows, this is found in the directory, *ORACLE_HOME*`\Apache\modplsql\conf`.

Note: The file `cache.README` in the `conf` directory includes a full description of each parameter, and provides examples showing how to set parameter values.

Table 4–7 mod_plsql cache.conf Configuration Parameter Summary

Parameter	Description
`PlsqlCacheCleanupTime`	Sets the interval for running cache cleanup routines. Default: `Everyday 23:00` (run cleanup routine daily at 11PM local time) **See Also:** Section 4.6.5, "Configuring Cache Cleanup"
`PlsqlCacheDirectory`	Defines the directory that holds the mod_plsql cache. Default: On UNIX systems, the default directory for the error log is: *ORACLE_HOME*`/Apache/modplsql/cache` On Windows systems, the default directory is: *ORACLE_HOME*`\Apache\modplsql\cache` **See Also:** Section 4.6.3, "Configuring File System Cache to Reside on a Faster File System"
`PlsqlCacheEnable`	Enables the file system cache. Default: `On` **See Also:** Section 4.6.2, "Enabling File System Cache"

Table 4–7 (Cont.) mod_plsql cache.conf Configuration Parameter Summary

Parameter	Description
`PlsqlCacheMaxAge`	Controls the aging, in days for the cache contents. Default: `30` (days) **See Also:** Section 4.6.4.2, "Setting the Days of Aging for Cache with PlsqlCacheMaxAge"
`PlsqlCacheMaxSize`	Sets the maximum size, in bytes, for an individual file stored in the cache. Default: `1048576` (1 Megabyte) **See Also:** Section 4.6.4.3, "Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize"
`PlsqlCacheTotalSize`	Limits the total size of the cache. The value is specified in bytes. Default: `20971520` (20 Megabytes) **See Also:** Section 4.6.4, "Resizing File System Cache"

4.6.2 Enabling File System Cache

The `cache.conf` parameter `PlsqlCacheEnable` enables mod_plsql caching. For maximum performance, enable `PlsqlCacheEnable` by setting the value of this parameter to `On`.

4.6.3 Configuring File System Cache to Reside on a Faster File System

This section describes how to configure a File System Cache to reside on a separate disk. When you use File System Cache and store the cache on a faster separate disk, performance should improve for all types of Web applications using File System Cache, including OracleAS Portal and generic PL/SQL based Web applications.

When you configure File System Cache, the cache can reside either on a separate physical disk or in a memory disk.

To set up a File System Cache on a separate disk:

1. Assume that the file system for the cache resides at the location:

   On UNIX: `/u01/cache`

   On Windows: `E:\cache`

2. Update the file:

   On UNIX: *`ORACLE_HOME`*`/Apache/modplsql/conf/cache.conf`

   On Windows: *`ORACLE_HOME`*`\Apache\modplsql\conf\cache.conf`

3. Change the cache parameter `PlsqlCacheDirectory`:

   On UNIX: `PlsqlCacheDirectory /u01/cache`

   On Windows: `PlsqlCacheDirectory  E:\cache`

4.6.4 Resizing File System Cache

This section covers the following topics:

- Setting the Total Cache Size with PlsqlCacheTotalSize
- Setting the Days of Aging for Cache with PlsqlCacheMaxAge
- Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize

4.6.4.1 Setting the Total Cache Size with PlsqlCacheTotalSize

The default installation sets the mod_plsql file system cache size to 2097152 bytes (20 Megabytes). If your PL/SQL application does not make use of the OWA_CACHE packages, or uses them to cache small amounts of content, then the default setting should be sufficient. If your PL/SQL application caches a lot of content in the mod_plsql file system cache, you should consider specifying a higher value.

To control the cache size, set the `PlsqlCacheTotalSize` parameter in the file `cache.conf`. On UNIX systems, this file is located under *`ORACLE_HOME`*`/Apache/modplsql/conf` directory. On Windows systems, this file is located under *`ORACLE_HOME`*`\Apache\modplsql\conf`.

You need to set the cache size high enough to achieve a high cache hit ratio. Try to set the cache size large enough so that frequently accessed content stays cached. It is also important to limit the amount of disk space, so that the cache size does not grow too large. Correct tuning for the cache size provides enough cache to hold all frequently accessed content while preventing the cache size from growing too large, since a very large cache is inefficient to search.

The value for `PlsqlCacheTotalSize` is specified as a number of bytes. 1MB equals 1048576 bytes. This setting is a soft limit on the amount of cache allocated. In some cases, the cache size may grow beyond this limit until the next cleanup operation. Therefore, the hard limit on the cache size is the underlying physical hard disk size. When this limit is reached, no cache content can be written out to disk until space is available.

To determine a reasonable cache size, do the following:

1. Turn on mod_plsql performance logging by setting the `LogLevel` in `httpd.conf` to the `info` level to enable mod_plsql logging.
2. Monitor the `error_log` on a daily basis. On UNIX systems, the default directory for the error log is: *`ORACLE_HOME`*`/Apache/Apache/log`. On Windows systems, the default directory is: *`ORACLE_HOME`*`\Apache\Apache\log`.

The mod_plsql `error_log` entries have the form:

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

where:

`deleted` is the number of cache files that got deleted during the cleanup process.

`max_age` is the number of cache files and total size that got deleted because they haven't been used for some time.

`kept` is the number of cache files and total size that was kept after the cleanup process.

`time` is the amount of time to perform the cleanup.

`limit` is the total cache size. This is the value of the `PlsqlCacheTotalSize` setting.

Interpret the entries in the error log as follows:

- If a high number of files are being deleted when compared to the number of files that were kept, this is a clear indication that your cache size is too small. You probably need to increase the size of the cache.
- If a low number of files being deleted when compared to the number of files that were kept is observed, this is an indication that your cache size is probably too big.

If you have enough disk space, you can chose to leave it as it or you can decrease the size of the cache to reclaim some disk space.

4.6.4.2 Setting the Days of Aging for Cache with PlsqlCacheMaxAge

Using the `PlsqlCacheMaxAge` parameter, you can control the "staleness" of cache content. The value for parameter is specified in units of days. The default value for this parameter is 30 (days). This means cache content is kept in the cache if it is less than 30 days old. After 30 days, the content is considered for deletion during the cleanup process.

The `max_age` information in mod_plsql `error_log` shows cache file aging information. If your site is a highly dynamic site, it would make sense to configure this setting to a lower value, since the older cache content will usually not be used again and, therefore, the lower value does not affect the cache hit ratio. If the site contains many static pages, it would make sense to increase the value of `PlsqlCacheMaxAge` so that the cleanup process does not deliberately delete the cache content.

4.6.4.3 Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize

Using the `PlsqlCacheMaxSize` parameter, you can specify the maximum size for individual files in the cache. Using this parameter prevents the case in which one cache file fills up the entire cache.

The default value for this parameter is 1048576 (bytes). In general, set this parameter to a value that represents about 1-3% of the total cache size.

4.6.5 Configuring Cache Cleanup

The cache cleanup parameter determines the frequency in which the File System Cache is examined and, if necessary, cleaned up. The cache cleanup parameter, `PlsqlCacheCleanupTime` is specified in the `cache.conf` file. The frequency can be set to daily, weekly, or monthly. When specifying weekly cleanup, it is possible to specify the day of the week and the time of the day.

The default mod_plsql setting of `PlsqlCacheCleanupTime` is daily at 11PM local time. Therefore, by default, every night at 11PM, the cleanup routine runs. When you select the monthly frequency, the cleanup occurs on the first Saturday of each month.

Configuring this parameter correctly is important since cleaning up too often can result in a lower cache hit ratio and when cleaning does not occur often enough, the cache's disk usage may be excessive.

Monitor the cleanup activities using the entries in the mod_plsql `error_log`; then tune the cleanup parameter, `PlsqlCacheCleanupTime` by analyzing the entries.

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

Note the following:

- Seeing a large number for the cleanup time can be an indication that the cleanup frequency is set too low. When the log indicates that the cleanup operation is busy examining or deleting many cache files, increasing the cleanup frequency should decrease the time spent in the cleanup operation.
- If a high number of files are being deleted during the cleanup operation because of "staleness", this is an indication that the cleanup frequency is too low. In this case, increase the frequency so that the cleanup can actively delete "stale" cache content more frequently.

4.7 Oracle HTTP Server Directives

To improve PL/SQL performance in Oracle HTTP Server, you need to tune the Oracle HTTP Server directives appropriately for your configuration.

See Also: *Oracle HTTP Server Administrator's Guide*

A

Frequently Asked Questions

- What is mod_plsql?
- What is the PL/SQL Web Toolkit?
- How do I find the version of mod_plsql?
- How do I find the version of the OWA packages?
- How do I install the OWA packages?
- How do I uninstall the OWA packages?
- How do I detect and clean up duplicate OWA packages installed in the database?
- I am getting HTTP error codes while accessing PL/SQL procedures through mod_plsql.
- All my PL/SQL procedures return a "Document contains no data" error in Netscape, or a blank page in Internet Explorer.
- I have a performant PL/SQL procedure, but some of my HTTP requests through mod_plsql take more than 15 seconds.
- Can I use mod_plsql to run applications on my own database?
- How do I create a DAD for mod_plsql?
- What authentication modes are available in mod_plsql?
- What is the mod_plsql Cleanup Thread?
- What kind of database connection pooling is present in mod_plsql?
- How does mod_plsql clean up database sessions?
- What happens when pooled database connections exist in mod_plsql and the database is restarted?
- How does mod_plsql clean up cached content in the file system?
- Can I invoke mod_plsql without a "/pls" prefix in the URL?
- How can I improve PL/SQL and mod_plsql performance?
- What kinds of logging facilities are available in mod_plsql?
- What considerations should I have in mod_plsql for High Availability?
- What considerations should I have in mod_plsql when the database is separated by a firewall?
- How do I assert a different hostname, port, or request_protocol to the PL/SQL application?

- How do I disable access to procedure names that have a specific pattern?
- I see the error "HTTP-503 ORA-12154" in the file ORACLE_HOME/Apache/Apache/conf/error_log. What does this mean?

What is mod_plsql?

mod_plsql is an Oracle HTTP Server plug-in that communicates with the database by mapping browser requests into database stored procedure calls over a SQL*Net connection. It is generally indicated by a `/pls` virtual path. The mod_plsql gateway provides support for building and deploying PL/SQL-based applications on the Web. PL/SQL stored procedures can retrieve data from database tables and generate HTTP responses containing formatted data and HTML code to display in a Web browser. See the *Oracle Database Application Developer's Guide - Fundamentals* for more information.

What is the PL/SQL Web Toolkit?

The PL/SQL Web Toolkit enables you to develop Web applications as PL/SQL procedures stored in an Oracle database server. Packages in the toolkit define procedures, functions, and data types that you can use in your stored procedures. See the *Oracle Database Application Developer's Guide - Fundamentals* for more information.

How do I find the version of mod_plsql?

You can determine the version of mod_plsql by executing the `oversioncheck` script on the mod_plsql binary.

On UNIX platforms, issue the following command:

```
ORACLE_HOME/Apache/Apache/bin/oversioncheck ORACLE_HOME/Apache/modplsql/bin/modplsql.so
```

On Windows platforms, issue the following command:

```
ORACLE_HOME\Apache\Apache\bin\oversioncheck ORACLE_HOME\bin\modplsql.dll
```

How do I find the version of the OWA packages?

1. Use SQL*Plus and connect as any user to the database.
2. Execute the following command:

   ```
   select owa_util.get_version from dual;
   ```

   This shows the version of the OWA packages. For example, 9.0.4.0.1.

If this query fails, you are having an old version of OWA packages that does not have versioning. It is recommended that you upgrade to a newer version.

How do I install the OWA packages?

See Section 2.2, "Installing Required Packages" for more information.

How do I uninstall the OWA packages?

OWA packages are uninstalled by performing the following tasks:

1. Navigate to the directory from where the OWA packages were installed. For example:

   ```
   cd ORACLE_HOME/Apache/modplsql
   ```

2. Use SQL*Plus to connect as the owner of the OWA packages. This user should be the SYS user, unless you have an old version of the OWA packages.

3. Invoke the script `owadins.sql` to uninstall the OWA packages.

How do I detect and clean up duplicate OWA packages installed in the database?

The following SQL query is used to determine the location of the OWA packages:

```
SELECT OWNER, OBJECT_TYPE
FROM   DBA_OBJECTS
WHERE  OBJECT_NAME = 'OWA'
```

You will see the following results:

```
SQL>

1  SELECT OWNER, OBJECT_TYPE
2  FROM DBA_OBJECTS
3* WHERE OBJECT_NAME = 'OWA'

OWNER  OBJECT_TYPE
-----  -----------
SYS    PACKAGE
SYS    PACKAGE BODY
PUBLIC SYNONYM
```

If you see more lines than shown in the preceding SQL query, it means that older OWA packages exist in other schemas, which may cause issues for mod_plsql users. In such situations, uninstall all versions of the OWA packages from the database, and reinstall the OWA packages that ship with the product.

I am getting HTTP error codes while accessing PL/SQL procedures through mod_plsql.

mod_plsql logs detailed error messages to the Oracle HTTP Server file *ORACLE_HOME*`/Apache/Apache/logs/error_log`. Scan this file to understand the problem. For more information on mod_plsql logging, see "What kinds of logging facilities are available in mod_plsql?".

All my PL/SQL procedures return a "Document contains no data" error in Netscape, or a blank page in Internet Explorer.

This problem could occur if you have duplicate OWA Packages installed in the database. See "How do I detect and clean up duplicate OWA packages installed in the database?" for more information.

I have a performant PL/SQL procedure, but some of my HTTP requests through mod_plsql take more than 15 seconds.

The most common reason for this problem is that the middle-tier character set does not match that of the back-end database, and HTTP `KeepAlive` is enabled in Oracle HTTP Server. This kind of misconfiguration causes an invalid *Content-Length* to be sent back to the browser, causing the browser to detect the end of the response stream only when the `KeepAliveTimeout` interval causes the stream to be closed. To solve the problem, ensure that the `PlsqlNLSLanguage` parameter in the DAD matches that of the database.

Can I use mod_plsql to run applications on my own database?

Yes. But before you run your applications, you need to install the OWA packages into your database. See Section 2.2, "Installing Required Packages".

How do I create a DAD for mod_plsql?

Refer to the mod_plsql section in the *Oracle HTTP Server Administrator's Guide*.

What authentication modes are available in mod_plsql?

See Chapter 3, "Securing Application Database Access Through mod_plsql".

What is the mod_plsql Cleanup Thread?

mod_plsql starts a thread in each HTTPD process. The job of this thread is to clean up idle database sessions and the file system cache. This thread is called the **Cleanup Thread**.

What kind of database connection pooling is present in mod_plsql?

Refer to Chapter 4, "Optimizing PL/SQL Performance".

How does mod_plsql clean up database sessions?

mod_plsql cleans up unused database sessions based on the configuration setting of `PlsqlIdleSessionCleanupInterval`. Besides this, the configuration directive `PlsqlMaxRequestsPerSession` governs how many requests will be serviced from a pooled database session. Finally, database sessions are closed when HTTPD processes are shut down.

What happens when pooled database connections exist in mod_plsql and the database is restarted?

When the database connection is severed, the first request that attempts to execute a PL/SQL procedure using the severed connection will fail. Subsequent requests will reestablish a database session and start functioning normally. The number of failures will be directly proportional to the number of pooled database sessions. Future versions of mod_plsql will detect terminated connections automatically.

Note: If the database is not restarted within the time interval of `PlsqlIdleSessionCleanupInterval`, then the cleanup thread will clean up the severed sessions, and no errors will be seen by end-users.

How does mod_plsql clean up cached content in the file system?

The cleanup thread scans the file system cache based on the configuration of `PlsqlCacheCleanupTime`. The default cleanup time is 11 P.M. daily, local time.

Can I invoke mod_plsql without a "/pls" prefix in the URL?

Yes. Since mod_plsql uses the Oracle HTTP Server' `Location` directive, you can configure any virtual path to be serviced by mod_plsql.

How can I improve PL/SQL and mod_plsql performance?

Refer to Chapter 4, "Optimizing PL/SQL Performance".

What kinds of logging facilities are available in mod_plsql?

- By default, mod_plsql logs alerts, warnings, and errors to the Oracle HTTP Server `error_log` file *`ORACLE_HOME`*`/Apache/Apache/logs/error_log`. The amount of information logged by mod_plsql is controlled by the setting of Oracle

HTTP Server's `LogLevel` parameter in `httpd.conf`. By default, this is configured to `warn`.

- You can also enable performance logging for mod_plsql for every request as follows:

  1. Edit *ORACLE_HOME*`/Apache/Apache/conf/httpd.conf` and set `LogLevel` to `info` (default is `warn`).

  2. Restart Oracle HTTP Server using the following command:

     ```
     ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
     ```

  3. Issue some URLs to mod_plsql and verify that the file *ORACLE_HOME*`/Apache/Apache/logs/error_log` starts showing entries as follows:

     ```
     [Tue Apr 01 14:54:49 2003] [info] mod_plsql: [perf] 130.35.92.145
     /pls/app/htp.p status=200 user=scott reqTime=21ms connSU=(null),0ms
     connRO=(null),0ms connNSSO=HIT,1ms procTime=17ms sessionTidyTime=0ms
     cache=(null) cookie=(null),0ms pageCalls=0,0ms bytes=5 describe=No,0ms
     streamTime=0ms pid=175 sessFile=(null) userFile=834\0855 sysFile=470\5949
     cacheLevel=(null) cacheTime=0ms dbProcTime=15ms
     id=1049237685:130.35.92.145:373:1 spid=(null) qs=(null) requestTrace=(null)
     cookieLen=0 cookieValue=(null) reqUserTime=16ms assertUser=(null)
     subid=(null) authLevel=(null) oraError=0
     ```

- Finally, you can enable debug logging in mod_plsql. This is the highest level of logging and is not recommended for active sites.

  WARNING: This mode of logging should be enabled only at the request of Oracle Support Services.

  In this mode, debug messages are logged to Oracle HTTP Server's `error_log` file and additional mod_plsql specific logs are created under *ORACLE_HOME*`/Apache/modplsql/logs`. Log location is configurable using the `PlsqlLogDirectory` directive in *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf`. To enable debug level logging, perform the following tasks:

  1. Edit *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf` and set `PlsqlLogEnable` to `On` (default is `Off`).

  2. Restart Oracle HTTP Server using the following command:

     ```
     ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
     ```

What considerations should I have in mod_plsql for High Availability?

For high availability, mod_plsql based applications should be aware of the following things:

- The mod_plsql configuration parameter `PlsqlDatabaseConnectString` should use a connect string format of `NetServiceNameFormat` so that name resolution happens through an LDAP lookup of Oracle Internet Directory. This enables you to configure the database *host:port:service_name* information in a

central repository, which makes it easier to add or remove RAC nodes when required.

- mod_plsql does not automatically detect terminated database connections. In case a back end database goes down, the initial few requests fail. See "What happens when pooled database connections exist in mod_plsql and the database is restarted?" for more information.

What considerations should I have in mod_plsql when the database is separated by a firewall?

If a firewall exists between the middle-tier running mod_plsql, and the back end database, the idle session cleanup interval in mod_plsql should be configured lower than the idle session cleanup interval of the firewall. This ensures that the firewall never closes a connection established by mod_plsql.

Note: mod_plsql idle session cleanup interval can be configured using the parameter `PlsqlIdleSessionCleanupInterval` in *`ORACLE_HOME`*`/Apache/modplsql/conf/plsql.conf`. The default value is 15 minutes.

How do I assert a different hostname, port, or request_protocol to the PL/SQL application?

- In situations where your Oracle HTTP Server instance is front-ended by OracleAS Web Cache or a load balancing router (LBR), there is a need to assert the hostname and port for the site to be that of the Web Cache or the LBR. In such situations, it is recommended that you use the Oracle HTTP Server configuration directives `ServerName` and `Port` to do the assertion. If for some reason, you do not wish to assert the hostname and port at the Oracle HTTP Server level, you can use the mod_plsql configuration directive `PlsqlCGIEnvironmentList` to assert a different hostname and port to only the PL/SQL applications running under mod_plsql. For example:
  - `PlsqlCGIEnvironmentList SERVER_NAME=lbr.us.oracle.com`

    Consider using Oracle HTTP Server `ServerName` directive in `httpd.conf` instead.
  - `PlsqlCGIEnvironmentList SERVER_PORT=9999`

    Consider using Oracle HTTP Server `Port` directive in `httpd.conf` instead.
  - `PlsqlCGIEnvironmentList HTTP_HOST=myservername.us.oracle.com:9999`

    Combination of `SERVER_NAME:SERVER_PORT`.
- Similarly, in cases where your site is accessed externally as an SSL, but is internally running in non-SSL mode (with an SSL accelerator in between), you might want to assert the `REQUEST_PROTOCOL` as `HTTPS` so that the PL/SQL application generates SSL links instead of non-SSL links. For example:

```
PlsqlCGIEnvironmentList REQUEST_PROTOCOL=https
```

How do I disable access to procedure names that have a specific pattern?

Please refer to the description of `PlsqlExclusionList` in Section 3.3.1, "Using the PlsqlExclusionList Directive in mod_plsql".

I see the error "HTTP-503 ORA-12154" in the file ORACLE_HOME/Apache/Apache/conf/error_log. What does this mean?

This error means that mod_plsql is unable to connect to the database.

Ensure that:

1. The database is up and running.
2. The username and password information in the DAD is correct.
3. The middle-tier is able to connect to the database using the `PlsqlDatabaseConnectString` parameter in the DAD.

In most situations, the problem occurs because SQL*Net is not able to resolve the connect string parameter using the configuration information under *ORACLE_HOME*`/network/admin`.

- For entries configured with *TNSFormat* or *NetServiceNameFormat*, validate the connect string information by using `tnsping dad_connect_string`. For example:

  ```
  tnsping "cn=iasdb,cn=oraclecontext"
  ```

  or

  ```
  tnsping iasdb.us.oracle.com
  ```

- For entries configured with *SIDFormat* and *ServiceNameFormat*, ensure that the hostname, port, and SID/service_name information match for the database listener. After verifying this, confirm that SQL*Plus can connect to the database using the DAD username, password, and connect string.

If this does not work, refer to the Oracle SQL*Net documentation on how to troubleshoot this further.

Index

Symbols

! character
  definition, 1-3
  flexible parameter passing, 1-7

Numerics

2 parameter
  flexible parameter passing, 1-7
4 parameter
  flexible parameter passing, 1-8

A

arrays, 1-7

B

BLOB
  direct download, 1-16
  document table definition, 1-10

C

caching
  expires technique, 4-15
  owa_cache packages, 4-18
  system-level, 4-17
  user-level, 4-17
  validation technique, 4-13
CGI
  environment variables, 1-18
client request, 1-1
content column, 1-11
content_type column, 1-11
cookie restrictions, 1-20
creating
  DAD, 2-3

D

DAD
  creating a, 2-3
  definition, 1-2
DAD_charset column, 1-11
direct access URLs, 1-17
document access path, 1-12
document table definition, 1-10
  old style, 1-11
document_path, 1-12
document_proc, 1-13
download, 1-9
downloading files, 1-15
DTD, 1-10
  old style, 1-11

E

entity tag caching method, 4-13
environment variables
  CGI, 1-17
expires caching technique, 4-15

F

file upload, 1-9, 1-13
  attributes, 1-15
  multiple files, 1-15
four parameter
  flexible parameter passing, 1-8

G

GET method, 1-4

H

head method, 1-4
HTTP HEAD requests, 1-4
HTTP server
  httpd process, 4-3
httpd
  HTTP server process, 4-3
httpd.conf
  directives
    MaxSpareServers, 4-9
    MinSpareServers, 4-9
    StartServers, 4-9

K

KeepAlive httpd.conf directive, 4-23

L

language parameter (nls_lang), 1-19
LAST_UPDATED column, 1-11
LONGRAW
  document table definition, 1-10

M

mime type, 1-15
mod_expires, 4-23
mod_plsql
  invoking, 1-3

N

nls_lang
  definition, 1-19

O

overloading, 1-6, 1-7
owa_cache package, 4-18
owa_util PL/SQL web toolkit package, 1-17
owaload.sql, 2-1

P

parameters
  flexible, 1-7
  large, 1-9
  MaxClients, 4-5
  MaxRequestsPerChild, 4-5
  MaxSpareServers, 4-5
  MinSpareServers, 4-5
  overloaded, 1-6
  passing, 1-5, 1-7
  PlsqlIdleSessionCleanupInterval, 4-5
  PlsqlMaxRequestsPerSession, 4-5
performance
  tuning
    expires caching, 4-15
    mod_expires, 4-23
    system-level caching, 4-17
    validation caching, 4-13
PL/SQL application
  creating a DAD for, 2-3
PL/SQL web toolkit functions, 4-18
POST method, 1-4

R

request_charset, 1-19
REQUEST_IANA_CHARSET, 1-20
restrictions, 1-20

S

system-level caching, 4-17

T

transaction model, 1-5
tuning
  expires caching technique, 4-15
  system-level caching, 4-17
  validation caching, 4-13
two parameter
  flexible parameter passing, 1-7

U

upload, 1-9
user-level caching, 4-17

V

validation caching
  for mod_plsql, 4-13
  technique, 4-13

W

web toolkit, 4-18

Exhibit 25

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 26

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 27

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 28

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 29

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 30

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 31

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 32

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY