Oracle9*i*

Supplied PL/SQL Packages and Types Reference

Release 2 (9.2)

March 2002

Part No. A96612-01

ORACLE

Oracle9*i* Supplied PL/SQL Packages and Types Reference, Release 2 (9.2)

Part No. A96612-01

Primary Author: D.K. Bradshaw

Contributing Authors: Ted Burroughs, Shelley Higgins, Paul Lane, Roza Leyderman, Kevin Macdowell, Jack Melnick, Chuck Murray, Kathy Rich, Vivian Schupmann, Randy Urbano

Contributors: D. Alpern, G. Arora, L. Barton, N. Bhatt, S. Chandrasekar, T. Chang, G. Claborn, R. Decker, A. Downing, J. Draaijer, S. Ehrsam, A. Ganesh, R. Govindarajan, B. Goyal, C. Iyer, H. Jakobsson, A. Kalra, B. Lee, J. Liu, P. Locke, A. Logan, V. Maganty, N. Mallavarupu, J. Mallory, R. Mani, S. Mavris, A. Mozes, J. Muller, K. Muthukkaruppan, R. Pang, D. Raphaely, S. Ray, A. Rhee, J. Sharma, R. Sujithan, A. Swaminathan, K. Tarkhanov, A. Tsukerman A. To, S. Urman, S. Vivian, D. Voss, W. Wang, D. Wong

Graphics Production Specialist: Valarie Moore

Contents

Send Us Your Comments

Oracle9*i* Supplied PL/SQL Packages and Types Reference, Release 2 (9.2)

Part No. A96612-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This reference manual describes the Oracle PL/SQL packages shipped with the Oracle database server. This information applies to versions of the Oracle database server that run on all platforms unless otherwise specified.

This preface contains these topics:

Audience

Oracle9i Supplied PL/SQL Packages and Types Reference is intended for programmers, systems analysts, project managers, and others interested in developing database applications. This manual assumes a working knowledge of application programming and familiarity with SQL to access information in relational database systems. Some sections also assume a knowledge of basic object-oriented programming.

Organization

See Table 1–1, " Summary of Oracle Supplied PL/SQL Packages" on page 1-7 for information about the organization of this reference.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Application Developer's Guide - Fundamentals*
- *PL/SQL User's Guide and Reference*
- *Oracle9i Supplied Java Packages Reference.*

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. The `JRepUtil` class implements these methods.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
`{ }`	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
`|`	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
`...`	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS `*`subquery`*`;` `SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`

Convention	Meaning	Example
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS may not always read a line of text that consists solely of a bracket or brace.

What's New in Supplied PL/SQL Packages and Types?

The following sections describe the new features in Oracle Supplied PL/SQL Packages and Types:

- Oracle9i Release 2 (9.2) Beta New Features in Supplied PL/SQL Packages and Types
- Oracle9i Release 1 (9.0.1) New Features in Supplied PL/SQL Packages and Types
- Oracle8i Release 2 (8.1.6) New Features in Supplied PL/SQL Packages
- Oracle8i Release 1 (8.1.5) New Features in Supplied PL/SQL Packages

Oracle9*i* Release 2 (9.2) Beta New Features in Supplied PL/SQL Packages and Types

This release includes the following new chapters:

- Advanced Queuing Types
- DBMS_APPLY_ADM
- DBMS_CAPTURE_ADM
- DBMS_LOGSTDBY
- DBMS_MGWADM
- DBMS_MGWMSG
- DBMS_PROPAGATION_ADM
- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_STORAGE_MAP
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_XDB
- DBMS_XDBT
- DBMS_XDB_VERSION
- DBMS_XMLDOM
- DBMS_XMLPARSER
- DBMS_XPLAN
- DBMS_XSLPROCESSOR
- JMS Types
- Logical Change Record Types
- Rule Types

This release includes changes to the following chapters:

- DBMS_DDL
- DBMS_FLASHBACK
- DBMS_LOB
- DBMS_LOGMNR
- DBMS_LOGMNR_CDC_PUBLISH
- DBMS_LOGMNR_CDC_SUBSCRIBE
- DBMS_LOGMNR_D
- DBMS_METADATA
- DBMS_REDEFINITION
- DBMS_RLS
- DBMS_SPACE_ADMIN
- DBMS_STATS
- DBMS_TRANSFORM
- DBMS_WM
- DBMS_XMLGEN
- DBMS_XMLQUERY
- DBMS_XMLSAVE
- DBMS_XMLSchema
- UTL_FILE
- UTL_HTTP

Oracle9*i* Release 1 (9.0.1) New Features in Supplied PL/SQL Packages and Types

This release includes the following new packages:

- DBMS_AQELM
- DBMS_ENCODE
- DBMS_FGA
- DBMS_FLASHBACK
- DBMS_LDAP
- DBMS_LibCache
- DBMS_LOGMNR_CDC_PUBLISH
- DBMS_LOGMNR_CDC_SUBSCRIBE
- DBMS_METADATA
- DBMS_ODCI
- DBMS_OUTLN_EDIT
- DBMS_REDEFINITION
- DBMS_TRANSFORM
- DBMS_URL
- DBMS_WM
- DBMS_XMLGEN
- DBMS_XMLQuery
- DMBS_XMLSave
- UTL_ENCODE

This release includes new information about types:

- DBMS_TYPES
- ANYDATA_TYPE
- ANYDATASET_TYPE
- ANYTYPE_TYPE

This release includes enhancements to the following packages:

- UTL_FILE
- UTL_HTTP
- UTL_RAW

Oracle8*i* Release 2 (8.1.6) New Features in Supplied PL/SQL Packages

This release included the following new packages

- DBMS_BACKUP_RESTORE
- DBMS_OBFUSCATION_TOOLKIT
- UTL_INADDR
- UTL_SMTP
- UTL_TCP

This release included enhancements to the following packages:

- DBMS_DEBUG
- DBMS_DISTRIBUTED_TRUST_ADMIN
- DBMS_LOGMINER
- DBMS_LOGMINER_D
- DBMS_PCLXUTIL
- DMBS_PROFILER
- DBMS_REPAIR
- DBMS_RESOURCE_MANAGER
- DBMS_ROWID
- DBMS_SQL
- DBMS_UTILITY
- UTL_HTTP

Oracle8*i* Release 1 (8.1.5) New Features in Supplied PL/SQL Packages

This book was new for release 8.1.5.

1

Introduction

Oracle supplies many PL/SQL packages with the Oracle server to extend database functionality and provide PL/SQL access to SQL features. You can use the supplied packages when creating your applications or for ideas in creating your own stored procedures.

> **Note:** This manual covers the packages provided with the Oracle database server. Packages supplied with other products, such as Oracle Developer or the Oracle Application Server, are not covered.

This chapter contains the following topics:

> **See Also:** *Oracle9i Application Developer's Guide - Fundamentals* for information on how to create your own packages

Package Overview

A *package* is an encapsulated collection of related program objects stored together in the database. Program objects are procedures, functions, variables, constants, cursors, and exceptions.

Packages have many advantages over standalone procedures and functions. For example, they:

- Let you organize your application development more efficiently.
- Let you grant privileges more efficiently.
- Let you modify package objects without recompiling dependent schema objects.
- Enable Oracle to read multiple package objects into memory at once.
- Let you *overload* procedures or functions. Overloading means creating multiple procedures with the same name in the same package, each taking arguments of different number or datatype.
- Can contain global variables and cursors that are available to all procedures and functions in the package.

Package Components

PL/SQL packages have two parts: the specification and the body, although sometimes the body is unnecessary. The specification is the interface to your application; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specification.

Unlike subprograms, packages cannot be called, parameterized, or nested. However, the formats of a package and a subprogram are similar:

```
CREATE PACKAGE name AS  -- specification (visible part)
   -- public type and item declarations
   -- subprogram specifications
END [name];

CREATE PACKAGE BODY name AS  -- body (hidden part)
   -- private type and item declarations
   -- subprogram bodies
[BEGIN
   -- initialization statements]
END [name];
```

The specification holds public declarations that are visible to your application. The body holds implementation details and private declarations that are hidden from your application. You can debug, enhance, or replace a package body without changing the specification. You can change a package body without recompiling calling programs because the implementation details in the body are hidden from your application.

Using Oracle Supplied Packages

Most Oracle supplied packages are automatically installed when the database is created and the CATPROC.SQL script is run. For example, to create the DBMS_ALERT package, the DBMSALRT.SQL and PRVTALRT.PLB scripts must be run when connected as the user SYS. These scripts are run automatically by the CATPROC.SQL script.

Certain packages are not installed automatically. Special installation instructions for these packages are documented in the individual chapters.

To call a PL/SQL function from SQL, you must either own the function or have EXECUTE privileges on the function. To select from a view defined with a PL/SQL function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view. Instructions on special requirements for packages are documented in the individual chapters.

Creating New Packages

To create packages and store them permanently in an Oracle database, use the CREATE PACKAGE and CREATE PACKAGE BODY statements. You can execute these statements interactively from SQL*Plus or Enterprise Manager.

To create a new package, do the following:

1. Create the package specification with the CREATE PACKAGE statement.

   You can declare program objects in the package specification. Such objects are called *public* objects. Public objects can be referenced outside the package, as well as by other objects in the package.

   Note: It is often more convenient to add the OR REPLACE clause in the CREATE PACKAGE statement.

2. Create the package body with the CREATE PACKAGE BODY statement.

You can declare and define program objects in the package body.

- You must define public objects declared in the package specification.
- You can declare and define additional package objects, called *private* objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package. They cannot be referenced outside the package.

See Also:

- *PL/SQL User's Guide and Reference*
- *Oracle9i Application Developer's Guide - Fundamentals*

for more information on creating new packages

- *Oracle9i Database Concepts*

for more information on storing and executing packages

Separating the Specification and Body

The specification of a package declares the public types, variables, constants, and subprograms that are visible outside the immediate scope of the package. The body of a package defines the objects declared in the specification, as well as private objects that are not visible to applications outside the package.

Oracle stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. Using this distinction, you can change the definition of a program object in the package body without causing Oracle to invalidate other schema objects that call or reference the program object. Oracle invalidates dependent schema objects only if you change the declaration of the program object in the package specification.

Example The following example shows a package specification for a package named EMPLOYEE_MANAGEMENT. The package contains one stored function and two stored procedures.

```
CREATE PACKAGE employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER;
   PROCEDURE fire_emp (emp_id NUMBER);
   PROCEDURE sal_raise (emp_id NUMBER, sal_incr NUMBER);
END employee_management;
```

The body for this package defines the function and the procedures:

```
CREATE PACKAGE BODY employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER IS
```

The function accepts all arguments for the fields in the employee table except for the employee number. A value for this field is supplied by a sequence. The function returns the sequence number generated by the call to this function.

```
       new_empno    NUMBER(10);

   BEGIN
      SELECT emp_sequence.NEXTVAL INTO new_empno FROM dual;
      INSERT INTO emp VALUES (new_empno, name, job, mgr,
         hiredate, sal, comm, deptno);
      RETURN (new_empno);
   END hire_emp;

   PROCEDURE fire_emp(emp_id IN NUMBER) AS
```

The procedure deletes the employee with an employee number that corresponds to the argument `emp_id`. If no employee is found, then an exception is raised.

```
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
      IF SQL%NOTFOUND THEN
      raise_application_error(-20011, 'Invalid Employee
         Number: ' || TO_CHAR(emp_id));
   END IF;
END fire_emp;

PROCEDURE sal_raise (emp_id IN NUMBER, sal_incr IN NUMBER) AS
```

The procedure accepts two arguments. `Emp_id` is a number that corresponds to an employee number. `Sal_incr` is the amount by which to increase the employee's salary.

```
   BEGIN

   -- If employee exists, then update salary with increase.

      UPDATE emp
         SET sal = sal + sal_incr
         WHERE empno = emp_id;
```

```
      IF SQL%NOTFOUND THEN
         raise_application_error(-20011, 'Invalid Employee
            Number: ' || TO_CHAR(emp_id));
      END IF;
   END sal_raise;
END employee_management;
```

Note: If you want to try this example, then first create the sequence number `emp_sequence`. You can do this using the following SQL*Plus statement:

```
SQL> CREATE SEQUENCE emp_sequence
   > START WITH 8000 INCREMENT BY 10;
```

Referencing Package Contents

To reference the types, items, and subprograms declared in a package specification, use the dot notation. For example:

```
package_name.type_name
package_name.item_name
package_name.subprogram_name
```

Abbreviations for Datetime and Interval Datatypes

Many of the datetime and interval datatypes have names that are too long to be used with the procedures and functions in the replication management API. Therefore, you must use abbreviations for these datatypes instead of the full names. The following table lists each datatype and its abbreviation. No abbreviation is necessary for the `DATE` and `TIMESTAMP` datatypes.

Datatype	Abbreviation
`TIMESTAMP WITH TIME ZONE`	`TSTZ`
`TIMESTAMP LOCAL TIME ZONE`	`TSLTZ`
`INTERVAL YEAR TO MONTH`	`IYM`
`INTERVAL DAY TO SECOND`	`IDS`

For example, if you want to use the `DBMS_DEFER_QUERY.GET_`*`datatype`*`_ARG` function to determine the value of a `TIMESTAMP LOCAL TIME ZONE` argument in a

deferred call, then you substitute TSLTZ for *datatype*. Therefore, you run the DBMS_DEFER_QUERY.GET_TSLTZ_ARG function.

Summary of Oracle Supplied PL/SQL Packages

Table 1–1 lists the supplied PL/SQL server packages. These packages run as the invoking user, rather than the package owner. Unless otherwise noted, the packages are callable through public synonyms of the same name.

Caution:

- The procedures and functions provided in these packages and their external interfaces are reserved by Oracle and are subject to change.
- Modifying Oracle supplied packages can cause internal errors and database security violations. Do not modify supplied packages.

Table 1–1 Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
CWM2_OLAP_AW_ACCESS	Generates scripts that create relational views of analytic workspace objects.	*Oracle9i OLAP User's Guide*
DBMS_ALERT	Provides support for the asynchronous notification of database events.	Chapter 2
DBMS_APPLICATION_INFO	Lets you register an application name with the database for auditing or performance tracking purposes.	Chapter 3
DBMS_APPLY_ADM	Provides administrative procedures to start, stop, and configure an apply process.	Chapter 4
DBMS_AQ	Lets you add a message (of a predefined object type) onto a queue or to dequeue a message.	Chapter 5
DBMS_AQADM	Lets you perform administrative functions on a queue or queue table for messages of a predefined object type.	Chapter 6
DBMS_AQELM	Provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP.	Chapter 7

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_AW	Issues OLAP DML statements against analytic workspace objects. Also, lets you retrieve and print the session logs created by the execution of the procedures and functions in this package and the OLAP_TABLE function.	*Oracle9i OLAP User's Guide*
DBMS_CAPTURE_ADM	Describes administrative procedures to start, stop, and configure a capture process; used in Streams.	Chapter 8
DBMS_DDL	Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs.	Chapter 9
DBMS_DEBUG	Implements server-side debuggers and provides a way to debug server-side PL/SQL program units.	Chapter 10
DBMS_DEFER	Provides the user interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.	Chapter 11
DBMS_DEFER_QUERY	Permits querying the deferred remote procedure calls (RPC) queue data that is not exposed through views. Requires the Distributed Option.	Chapter 12
DMBS_DEFER_SYS	Provides the system administrator interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.	Chapter 13
DBMS_DESCRIBE	Describes the arguments of a stored procedure with full name translation and security checking.	Chapter 14
DBMS_DISTRIBUTED_TRUST_ADMIN	Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted.	Chapter 15
DBMS_FGA	Provides fine-grained security functions.	Chapter 16
DMBS_FLASHBACK	Lets you flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN).	Chapter 17
DBMS_HS_PASSTHROUGH	Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems.	Chapter 18

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_IOT	Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command.	Chapter 19
DBMS_JOB	Lets you schedule administrative procedures that you want performed at periodic intervals; it is also the interface for the job queue.	Chapter 20
DBMS_LDAP	Provides functions and procedures to access data from LDAP servers.	Chapter 21
DBMS_LIBCACHE	Prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution.	Chapter 22
DBMS_LOB	Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read/write), and BFILEs (read-only).	Chapter 23
DBMS_LOCK	Lets you request, convert and release locks through Oracle Lock Management services.	Chapter 24
DBMS_LOGMNR	Provides functions to initialize and run the log reader.	Chapter 25
DBMS_LOGMNR_CDC_PUBLISH	Identifies new data that has been added to, modified, or removed from, relational tables and publishes the changed data in a form that is usable by an application.	Chapter 26
DBMS_LOGMNR_CDC_SUBSCRIBE	Lets you view and query the change data that was captured and published with the DBMS_LOGMNR_CDC_PUBLISH package.	Chapter 27
DBMS_LOGMNR_D	Queries the dictionary tables of the current database, and creates a text based file containing their contents.	Chapter 28
DBMS_LOGSTDBY	Describes procedures for configuring and managing the logical standby database environment.	Chapter 29
DBMS_METADATA	Lets callers easily retrieve complete database object definitions (metadata) from the dictionary.	Chapter 30
DBMS_MGWADM	Describes the Messaging Gateway administrative interface; used in Advanced Queuing.	Chapter 31

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_MGWMSG	Describes object types—used by the canonical message types to convert message bodies—and helper methods, constants, and subprograms for working with the Messaging Gateway message types; used in Advanced Queuing.	Chapter 32
DBMS_MVIEW	Lets you refresh snapshots that are not part of the same refresh group and purge logs. DBMS_SNAPSHOT is a synonym.	Chapter 33
DBMS_OBFUSCATION_TOOLKIT	Provides procedures for Data Encryption Standards.	Chapter 34
DBMS_ODCI	Returns the CPU cost of a user function based on the elapsed time of the function.	Chapter 35
DBMS_OFFLINE_OG	Provides public APIs for offline instantiation of master groups.	Chapter 36
DBMS_OFFLINE_SNAPSHOT	Provides public APIs for offline instantiation of snapshots.	Chapter 37
DBMS_OLAP	Provides procedures for summaries, dimensions, and query rewrites.	Chapter 38
DBMS_ORACLE_TRACE_AGENT	Provides client callable interfaces to the Oracle TRACE instrumentation within the Oracle7 Server.	Chapter 39
DBMS_ORACLE_TRACE_USER	Provides public access to the Oracle release 7 Server Oracle TRACE instrumentation for the calling user.	Chapter 40
DBMS_OUTLN	Provides the interface for procedures and functions associated with management of stored outlines. Synonymous with OUTLN_PKG	Chapter 41
DBMS_OUTLN_EDIT	Lets you edit an invoker's rights package.	Chapter 42
DBMS_OUTPUT	Accumulates information in a buffer so that it can be retrieved out later.	Chapter 43
DBMS_PCLXUTIL	Provides intra-partition parallelism for creating partition-wise local indexes.	Chapter 44
DBMS_PIPE	Provides a DBMS pipe service which enables messages to be sent between sessions.	Chapter 45

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_PROFILER	Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks.	Chapter 46
DBMS_PROPAGATION_ADM	Provides administrative procedures for configuring propagation from a source queue to a destination queue.	Chapter 47
DBMS_RANDOM	Provides a built-in random number generator.	Chapter 48
DBMS_RECTIFIER_DIFF	Provides APIs used to detect and resolve data inconsistencies between two replicated sites.	Chapter 49
DBMS_REDEFINITION	Lets you perform an online reorganization of tables.	Chapter 50
DBMS_REFRESH	Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time. Requires the Distributed Option.	Chapter 51
DBMS_REPAIR	Provides data corruption repair procedures.	Chapter 52
DBMS_REPCAT	Provides routines to administer and update the replication catalog and environment. Requires the Replication Option.	Chapter 53
DBMS_REPCAT_ADMIN	Lets you create users with the privileges needed by the symmetric replication facility. Requires the Replication Option.	Chapter 54
DBMS_REPCAT_INSTATIATE	Instantiates deployment templates. Requires the Replication Option.	Chapter 55
DBMS_REPCAT_RGT	Controls the maintenance and definition of refresh group templates. Requires the Replication Option.	Chapter 56
DBMS_REPUTIL	Provides routines to generate shadow tables, triggers, and packages for table replication.	Chapter 57
DBMS_RESOURCE_MANAGER	Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema.	Chapter 58
DBMS_RESOURCE_MANAGER_PRIVS	Maintains privileges associated with resource consumer groups.	Chapter 59

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_RESUMABLE	Lets you suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution.	Chapter 60
DBMS_RLS	Provides row level security administrative interface.	Chapter 61
DBMS_ROWID	Provides procedures to create rowids and to interpret their contents.	Chapter 62
DBMS_RULE	Describes the EVALUATE procedure used in Streams.	Chapter 63
DBMS_RULE_ADM	Describes the administrative interface for creating and managing rules, rule sets, and rule evaluation contexts; used in Streams.	Chapter 64
DBMS_SESSION	Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures.	Chapter 65
DBMS_SHARED_POOL	Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism.	Chapter 66
DBMS_SPACE	Provides segment space information not available through standard SQL.	Chapter 67
DBMS_SPACE_ADMIN	Provides tablespace and segment space administration not available through the standard SQL.	Chapter 68
DBMS_SQL	Lets you use dynamic SQL to access the database.	Chapter 69
DBMS_STATS	Provides a mechanism for users to view and modify optimizer statistics gathered for database objects.	Chapter 70
DBMS_STORAGE_MAP	Communicates with FMON to invoke mapping operations.	Chapter 71
DBMS_STRM	Describes the interface to convert SYS.AnyData objects into LCR objects and an interface to annotate redo entries generated by a session with a binary tag.	Chapter 72

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_STRM_A	Describes administrative procedures for adding and removing simple rules, without transformations, for capture, propagation, and apply at the table, schema, and database level.	Chapter 73
DBMS_TRACE	Provides routines to start and stop PL/SQL tracing.	Chapter 74
DBMS_TRANSACTION	Provides access to SQL transaction statements from stored procedures and monitors transaction activities.	Chapter 75
DBMS_TRANSFORM	Provides an interface to the message format transformation features of Oracle Advanced Queuing.	Chapter 76
DBMS_TTS	Checks if the transportable set is self-contained.	Chapter 77
DBMS_TYPES	Consists of constants, which represent the built-in and user-defined types.	Chapter 78
DBMS_UTILITY	Provides various utility routines.	Chapter 79
DBMS_WM	Describes how to use the programming interface to Oracle Database Workspace Manager to work with long transactions.	Chapter 80
DBMS_XDB	Describes Resource Management and Access Control APIs for PL/SQL	Chapter 81
DBMS_XDBT	Describes how an administrator can create a ConText index on the XML DB hierarchy and configure it for automatic maintenance	Chapter 82
DBMS_XDB_VERSION	Describes versioning APIs	Chapter 83
DBMS_XMLDOM	Explains access to XMLType objects	Chapter 84
DBMS_XMLGEN	Converts the results of a SQL query to a canonical XML format.	Chapter 85
DBMS_XMLPARSER	Explains access to the contents and structure of XML documents.	Chapter 86
DMBS_XMLQUERY	Provides database-to-XMLType functionality.	Chapter 87
DBMS_XMLSAVE	Provides XML-to-database-type functionality.	Chapter 88
DBMS_XMLSCHEMA	Explains procedures to register and delete XML schemas.	Chapter 89

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
DBMS_XPLAN	Describes how to format the output of the EXPLAIN PLAN command.	Chapter 90
DBMS_XSLPROCESSOR	Explains access to the contents and structure of XML documents.	Chapter 91
DEBUG_EXTPROC	Lets you debug external procedures on platforms with debuggers that attach to a running process.	Chapter 92
SDO_CS (refer to Note #1)	Provides functions for coordinate system transformation.	*Oracle Spatial User's Guide and Reference*
SDO_GEOM (refer to Note #1)	Provides functions implementing geometric operations on spatial objects.	*Oracle Spatial User's Guide and Reference*
SDO_LRS (refer to Note #1)	Provides functions for linear referencing system support.	*Oracle Spatial User's Guide and Reference*
SDO_MIGRATE (refer to Note #1)	Provides functions for migrating spatial data from previous releases.	*Oracle Spatial User's Guide and Reference*
SDO_TUNE (refer to Note #1)	Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in Oracle Spatial.	*Oracle Spatial User's Guide and Reference*
SDO_UTIL (refer to Note #1)	Provides utility functions and procedures for Oracle Spatial.	*Oracle Spatial User's Guide and Reference*
UTL_COLL	Enables PL/SQL programs to use collection locators to query and update.	Chapter 93
UTL_ENCODE	Provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts.	Chapter 94
UTL_FILE	Enables your PL/SQL programs to read and write operating system text files and provides a restricted version of standard operating system stream file I/O.	Chapter 95
UTL_HTTP	Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges.	Chapter 96
UTL_INADDR	Provides a procedure to support internet addressing.	Chapter 97

Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

Package Name	Description	Documentation
UTL_PG	Provides functions for converting COBOL numeric data into Oracle numbers and Oracle numbers into COBOL numeric data.	*Oracle Procedural Gateway for APPC User's Guide*
UTL_RAW	Provides SQL functions for RAW datatypes that concat, substr to and from RAWS.	Chapter 98
UTL_REF	Enables a PL/SQL program to access an object by providing a reference to the object.	Chapter 99
UTL_SMTP	Provides PL/SQL functionality to send emails.	Chapter 100
UTL_TCP	Provides PL/SQL functionality to support simple TCP/IP-based communications between servers and the outside world.	Chapter 101
UTL_URL	Provides escape and unescape mechanisms for URL characters.	Chapter 102
ANYDATA TYPE	A self-describing data instance type containing an instance of the type plus a description	Chapter 103
ANYDATASET TYPE	Contains a description of a given type plus a set of data instances of that type	Chapter 104
ANYTYPE TYPE	Contains a type description of any persistent SQL type, named or unnamed, including object types and collection types; or, it can be used to construct new transient type descriptions	Chapter 105
JMS TYPES	Describes JMS types so that a PL/SQL application can use JMS queues of JMS types	Chapter 107
ADVANCED QUEUING TYPES	Describes the types used in Advanced Queuing	Chapter 106
LOGICAL CHANGE RECORD TYPES	Describes LCR types, which are message payloads that contain information about changes to a database, used in Streams	Chapter 108
RULES TYPES	Describes the types used with rules, rule sets, and evaluation contexts	Chapter 109

Note #1

Spatial packages are installed in user MDSYS with public synonyms.

Summary of Subprograms in Supplemental Packages

The packages listed in this section are documented in other Oracle books. See Table 1–1 for the documentation reference for each package. See Table 1–2 through Table 1–8 for the subprograms provided with these packages.

SDO_CS Package

Table 1–2 SDO_CS Package Subprograms

Subprogram	Description
SDO_CS.TRANSFORM	Transforms a geometry representation using a coordinate system (specified by SRID or name).
SDO_CS.TRANSFORM_LAYER	Transforms an entire layer of geometries (that is, all geometries in a specified column in a table).
VIEWPORT_TRANSFORM	Transforms an optimized rectangle into a valid geodetic polygon for use with Spatial operators and functions.

SDO_GEOM Package

Table 1–3 SDO_GEOM Package Subprograms

Subprogram	Description
RELATE	Determines how two objects interact.
SDO_ARC_DENSIFY	Changes each circular arc into an approximation consisting of straight lines, and each circle into a polygon consisting of a series of straight lines that approximate the circle.
SDO_AREA	Computes the area of a two-dimensional polygon.
SDO_BUFFER	Generates a buffer polygon around a geometry.
SDO_CENTROID	Returns the centroid of a polygon.
SDO_CONVEXHULL	Returns a polygon-type object that represents the convex hull of a geometry object.
SDO_DIFFERENCE	Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.
SDO_DISTANCE	Computes the distance between two geometry objects.
SDO_INTERSECTION	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.

Table 1–3 (Cont.) SDO_GEOM Package Subprograms

Subprogram	Description
SDO_LENGTH	Computes the length or perimeter of a geometry.
SDO_MAX_MBR_ORDINATE	Returns the maximum value for the specified ordinate of the minimum bounding rectangle of a geometry object.
SDO_MBR	Returns the minimum bounding rectangle of a geometry.
SDO_MIN_MBR_ORDINATE	Returns the minimum value for the specified ordinate of the minimum bounding rectangle of a geometry object.
SDO_POINTONSURFACE	Returns a point that is guaranteed to be on the surface of a polygon.
SDO_UNION	Returns a geometry object that is the topological union (OR operation) of two geometry objects.
SDO_XOR	Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.
VALIDATE_GEOMETRY	Determines if a geometry is valid.
VALIDATE_GEOMETRY_WITH_CONTEXT	Performs a consistency check for valid geometry types and returns context information if the geometry is invalid. The function checks the representation of the geometry from the tables against the element definitions.
VALIDATE_LAYER	Determines if all the geometries stored in a column are valid.
VALIDATE_LAYER_WITH_CONTEXT	Examines a geometry column to determine if the stored geometries follow the defined rules for geometry objects, and returns context information about any invalid geometries.
WITHIN_DISTANCE	Determines if two geometries are within a specified Euclidean distance from one another.

SDO_LRS Package

Table 1–4 SDO_LRS Package Subprograms

Subprogram	Description
CLIP_GEOM_SEGMENT	Clips a geometric segment (synonym of DYNAMIC_SEGMENT).
CONCATENATE_GEOM_SEGMENTS	Concatenates two geometric segments into one segment.

Table 1–4 (Cont.) SDO_LRS Package Subprograms

Subprogram	Description
CONNECTED_GEOM_SEGMENTS	Checks if two geometric segments are connected.
CONVERT_TO_LRS_DIM_ARRAY	Converts a standard dimensional array to a Linear Referencing System dimensional array by creating a measure dimension.
CONVERT_TO_LRS_GEOM	Converts a standard SDO_GEOMETRY line string to a Linear Referencing System geometric segment by adding measure information.
CONVERT_TO_LRS_LAYER	Converts all geometry objects in a column of type SDO_GEOMETRY from standard line string geometries without measure information to Linear Referencing System geometric segments with measure information, and updates the metadata.
CONVERT_TO_STD_DIM_ARRAY	Converts a Linear Referencing System dimensional array to a standard dimensional array by removing the measure dimension.
CONVERT_TO_STD_GEOM	Converts a Linear Referencing System geometric segment to a standard SDO_ GEOMETRY line string by removing measure information.
CONVERT_TO_STD_LAYER	Converts all geometry objects in a column of type SDO_GEOMETRY from Linear Referencing System geometric segments with measure information to standard line string geometries without measure information, and updates the metadata.
DEFINE_GEOM_SEGMENT	Defines a geometric segment.
DYNAMIC_SEGMENT	Clips a geometric segment (synonym of CLIP_ GEOM_SEGMENT).
FIND_LRS_DIM_POS	Returns the position of the measure dimension within the SDO_DIM_ARRAY structure for a specified SDO_GEOMETRY column.
FIND_MEASURE	Returns the measure of the closest point on a segment to a specified projection point.
GEOM_SEGMENT_END_MEASURE	Returns the end measure of a geometric segment.

Table 1–4 (Cont.) SDO_LRS Package Subprograms

Subprogram	Description
GEOM_SEGMENT_END_PT	Returns the end point of a geometric segment.
GEOM_SEGMENT_LENGTH	Returns the length of a geometric segment.
GEOM_SEGMENT_START_MEASURE	Returns the start measure of a geometric segment.
GEOM_SEGMENT_START_PT	Returns the start point of a geometric segment.
GET_MEASURE	Returns the measure of an LRS point.
IS_GEOM_SEGMENT_DEFINED	Checks if an LRS segment is defined correctly.
IS_MEASURE_DECREASING	Checks if the measure values along an LRS segment are decreasing (that is, descending in numerical value).
IS_MEASURE_INCREASING	Checks if the measure values along an LRS segment are increasing (that is, ascending in numerical value).
LOCATE_PT	Returns the point located at a specified distance from the start of a geometric segment.
MEASURE_RANGE	Returns the measure range of a geometric segment, that is, the difference between the start measure and end measure.
MEASURE_TO_PERCENTAGE	Returns the percentage (0 to 100) that a specified measure is of the measure range of a geometric segment.
OFFSET_GEOM_SEGMENT	Returns the geometric segment at a specified offset from a geometric segment.
PERCENTAGE_TO_MEASURE	Returns the measure value of a specified percentage (0 to 100) of the measure range of a geometric segment.
PROJECT_PT	Returns the projection point of a point on a geometric segment.
REDEFINE_GEOM_SEGMENT	Populates the measures of all shape points of a geometric segment based on the start and end measures, overriding any previously assigned measures between the start point and end point.

Table 1–4 (Cont.) SDO_LRS Package Subprograms

Subprogram	Description
RESET_MEASURE	Sets all measures of a geometric segment, including the start and end measures, to null values, overriding any previously assigned measures.
REVERSE_GEOMETRY	Returns a new geometric segment by reversing the measure values and the direction of the original geometric segment.
REVERSE_MEASURE	Returns a new geometric segment by reversing the original geometric segment.
SCALE_GEOM_SEGMENT	Scales a geometric segment.
SET_PT_MEASURE	Sets the measure value of a specified point.
SPLIT_GEOM_SEGMENT	Splits a geometric segment into two segments.
TRANSLATE_MEASURE	Returns a new geometric segment by translating the original geometric segment (that is, shifting the start and end measures by a specified value).
VALID_GEOM_SEGMENT	Checks if a geometric segment is valid.
VALID_LRS_PT	Checks if an LRS point is valid.
VALID_MEASURE	Checks if a measure falls within the measure range of a geometric segment.
VALIDATE_LRS_GEOMETRY	Checks if an LRS geometry is valid.

SDO_MIGRATE Package

Table 1–5 SDO_MIGRATE Package Subprograms

Procedure	Description
FROM_815_TO_81X	Migrates data from Spatial release 8.1.5 to the current release.
OGIS_METADATA_FROM	Generates a temporary table used when migrating OGIS (OpenGIS) metadata tables.
OGIS_METADATA_TO	Reads a temporary table used when migrating OGIS metadata tables.
TO_734	Migrates data from a previous release of Spatial Data Option to release 7.3.4.

Table 1–5 (Cont.) SDO_MIGRATE Package Subprograms

Procedure	Description
TO_81X	Migrates tables from Spatial Data Option 7.3.4 or Spatial Cartridge 8.0.4 to Oracle Spatial.
TO_CURRENT	Migrates data from a previous Spatial release to the current release.

SDO_TUNE Package

Table 1–6 SDO_TUNE Package Subprograms

Subprogram	Description
ANALYZE_RTREE	Analyzes an R-tree index; generates statistics about the index use, and recommends a rebuild of the index if a rebuild would improve query performance significantly.
AVERAGE_MBR	Calculates the average minimum bounding rectangle for geometries in a layer.
ESTIMATE_INDEX_ PERFORMANCE	Estimates the spatial index selectivity.
ESTIMATE_TILING_LEVEL	Determines an appropriate tiling level for creating fixed-size index tiles.
ESTIMATE_TILING_TIME	Estimates the tiling time for a layer, in seconds.
ESTIMATE_TOTAL_NUMTILES	Estimates the total number of spatial tiles for a layer.
EXTENT_OF	Determines the minimum bounding rectangle of the data in a layer.
HISTOGRAM_ANALYSIS	Calculates statistical histograms for a spatial layer.
MIX_INFO	Calculates geometry type information for a spatial layer, such as the percentage of each geometry type.
QUALITY_DEGRADATION	Returns the quality degradation for an R-tree index or the average quality degradation for all index tables for an R-tree index.
RTREE_QUALITY	Returns the quality score for an R-tree index or the average quality score for all index tables for an R-tree index.

SDO_UTIL Package

Table 1–7 SDO_UTIL Package Subprograms

Subprogram	Description
EXTRACT	Returns the geometry that represents a specified element (and optionally a ring) of the input geometry.
GETVERTICES	Returns a table containing the coordinates of the vertices of the input geometry.

UTL_PG Package

Table 1–8 UTL_PG Package Subprograms

Subprogram	Description
MAKE_NUMBER_TO_RAW_FORMAT	Makes a number_to_raw format conversion specification used to convert an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format.
MAKE_RAW_TO_NUMBER_FORMAT	Makes a raw_to_number format conversion specification used to convert a RAW byte-string from the remote host internal format into an Oracle number of comparable precision and scale.
NUMBER_TO_RAW	Converts an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format.
NUMBER_TO_RAW_FORMAT	Converts, according to the number_to_raw conversion format n2rfmt, an Oracle number numval of declared precision and scale to a RAW byte-string in the remote host internal format.
RAW_TO_NUMBER	Converts a RAW byte-string from the remote host internal format into an Oracle number.
RAW_TO_NUMBER_FORMAT	Converts, according to the raw_to_number conversion format r2nfmt, a RAW byte-string rawval in the remote host internal format to an Oracle number.
WMSG	Extracts a warning message specified by wmsgitem from wmsgblk.
WMSGCNT	Tests a wmsgblk to determine how many warnings, if any, are present.

2

DBMS_ALERT

DBMS_ALERT supports asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can notify itself whenever values of interest in the database are changed.

For example, suppose a graphics tool is displaying a graph of some data from a database table. The graphics tool can, after reading and graphing the data, wait on a database alert (WAITONE) covering the data just read. The tool automatically wakes up when the data is changed by any other user. All that is required is that a trigger be placed on the database table, which performs a signal (SIGNAL) whenever the trigger is fired.

Alerts are transaction-based. This means that the waiting session is not alerted until the transaction signalling the alert commits. There can be any number of concurrent signalers of a given alert, and there can be any number of concurrent waiters on a given alert.

A waiting application is blocked in the database and cannot do any other work.

> **Note:** Because database alerters issue commits, they cannot be used with Oracle Forms. For more information on restrictions on calling stored procedures while Oracle Forms is active, refer to your Oracle Forms documentation.

This chapter discusses the following topics:

- Security, Constants, and Errors for DBMS_ALERT
- Using Alerts
- Summary of DBMS_ALERT Subprograms

Security, Constants, and Errors for DBMS_ALERT

Security

Security on this package can be controlled by granting EXECUTE on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. EXECUTE privilege on this cover package can then be granted rather than on this package.

Constants

```
maxwait constant integer :=  86400000; -- 1000 days
```

The maximum time to wait for an alert (this is essentially forever).

Errors

DBMS_ALERT raises the application error -20000 on error conditions. Table 2–1 shows the messages and the procedures that can raise them.

Table 2–1 DBMS_ALERT Error Messages

Error Message	Procedure
ORU-10001 lock request error, status: N	SIGNAL
ORU-10015 error: N waiting for pipe status	WAITANY
ORU-10016 error: N sending on pipe 'X'	SIGNAL
ORU-10017 error: N receiving on pipe 'X'	SIGNAL
ORU-10019 error: N on lock request	WAIT
ORU-10020 error: N on lock request	WAITANY
ORU-10021 lock request error; status: N	REGISTER
ORU-10022 lock request error, status: N	SIGNAL
ORU-10023 lock request error; status N	WAITONE
ORU-10024 there are no alerts registered	WAITANY
ORU-10025 lock request error; status N	REGISTER
ORU-10037 attempting to wait on uncommitted signal from same session	WAITONE

Using Alerts

The application can register for multiple events and can then wait for any of them to occur using the WAITANY procedure.

An application can also supply an optional timeout parameter to the WAITONE or WAITANY procedures. A timeout of 0 returns immediately if there is no pending alert.

The signalling session can optionally pass a message that is received by the waiting session.

Alerts can be signalled more often than the corresponding application wait calls. In such cases, the older alerts are discarded. The application always gets the latest alert (based on transaction commit times).

If the application does not require transaction-based alerts, the DBMS_PIPE package may provide a useful alternative.

See Also: Chapter 45, "DBMS_PIPE"

If the transaction is rolled back after the call to SIGNAL, no alert occurs.

It is possible to receive an alert, read the data, and find that no data has changed. This is because the data changed after the *prior* alert, but before the data was read for that *prior* alert.

Checking for Alerts

Usually, Oracle is event-driven; this means that there are no polling loops. There are two cases where polling loops can occur:

- Shared mode. If your database is running in shared mode, a polling loop is required to check for alerts from another instance. The polling loop defaults to one second and can be set by the SET_DEFAULTS procedure.
- WAITANY procedure. If you use the WAITANY procedure, and if a signalling session does a signal but does not commit within one second of the signal, a polling loop is required so that this uncommitted alert does not camouflage other alerts. The polling loop begins at a one second interval and exponentially backs off to 30-second intervals.

Summary of DBMS_ALERT Subprograms

Table 2–2 DBMS_ALERT Package Subprograms

Subprogram	Description
REGISTER Procedure on page 2-4	Receives messages from an alert.
REMOVE Procedure on page 2-5	Disables notification from an alert.
REMOVEALL Procedure on page 2-5	Removes all alerts for this session from the registration list.
SET_DEFAULTS Procedure on page 2-6	Sets the polling interval.
SIGNAL Procedure on page 2-6	Signals an alert (send message to registered sessions).
WAITANY Procedure on page 2-7	Waits `timeout` seconds to receive alert message from an alert registered for session.
WAITONE Procedure on page 2-8	Waits `timeout` seconds to receive message from named alert.

REGISTER Procedure

This procedure lets a session register interest in an alert. The name of the alert is the `IN` parameter. A session can register interest in an unlimited number of alerts. Alerts should be deregistered when the session no longer has any interest, by calling `REMOVE`.

Syntax

```
DBMS_ALERT.REGISTER (
   name  IN  VARCHAR2);
```

Parameters

Table 2–3 REGISTER Procedure Parameters

Parameter	Description
`name`	Name of the alert in which this session is interested.

Caution: Alert names beginning with 'ORA$' are reserved for use for products provided by Oracle Corporation. Names must be 30 bytes or less. The name is case insensitive.

REMOVE Procedure

This procedure enables a session that is no longer interested in an alert to remove that alert from its registration list. Removing an alert reduces the amount of work done by signalers of the alert.

Removing alerts is important because it reduces the amount of work done by signalers of the alert. If a session dies without removing the alert, that alert is eventually (but not immediately) cleaned up.

Syntax

```
DBMS_ALERT.REMOVE (
   name  IN  VARCHAR2);
```

Parameters

Table 2–4 REMOVE Procedure Parameters

Parameter	Description
name	Name of the alert (case-insensitive) to be removed from registration list.

REMOVEALL Procedure

This procedure removes all alerts for this session from the registration list. You should do this when the session is no longer interested in any alerts.

This procedure is called automatically upon first reference to this package during a session. Therefore, no alerts from prior sessions which may have terminated abnormally can affect this session.

This procedure always performs a commit.

Syntax

```
DBMS_ALERT.REMOVEALL;
```

SET_DEFAULTS Procedure

In case a polling loop is required, use the SET_DEFAULTS procedure to set the polling interval.

Syntax

```
DBMS_ALERT.SET_DEFAULTS (
   sensitivity  IN  NUMBER);
```

Parameters

Table 2–5 SET_DEFAULTS Procedure Parameters

Parameter	Description
sensitivity	Polling interval, in seconds, to sleep between polls. The default interval is five seconds.

SIGNAL Procedure

This procedure signals an alert. The effect of the SIGNAL call only occurs when the transaction in which it is made commits. If the transaction rolls back, SIGNAL has no effect.

All sessions that have registered interest in this alert are notified. If the interested sessions are currently waiting, they are awakened. If the interested sessions are not currently waiting, they are notified the next time they do a wait call.

Multiple sessions can concurrently perform signals on the same alert. Each session, as it signals the alert, blocks all other concurrent sessions until it commits. This has the effect of serializing the transactions.

Syntax

```
DBMS_ALERT.SIGNAL (
   name     IN  VARCHAR2,
   message  IN  VARCHAR2);
```

Parameters

Table 2–6 SIGNAL Procedure Parameters

Parameter	Description
name	Name of the alert to signal.
message	Message, of 1800 bytes or less, to associate with this alert. This message is passed to the waiting session. The waiting session might be able to avoid reading the database after the alert occurs by using the information in the message.

WAITANY Procedure

Call WAITANY to wait for an alert to occur for any of the alerts for which the current session is registered. An implicit COMMIT is issued before this procedure is executed. The same session that waits for the alert may also first signal the alert. In this case remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Syntax

```
DBMS_ALERT.WAITANY (
   name      OUT  VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 2–7 WAITANY Procedure Parameters

Parameter	Description
name	Returns the name of the alert that occurred.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITANY, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - time-out occurred
timeout	Maximum time to wait for an alert. If no alert occurs before timeout seconds, this returns a status of 1.

Errors

```
-20000, ORU-10024: there are no alerts registered.
```

Cause: You must register an alert before waiting.

WAITONE Procedure

This procedure waits for a specific alert to occur. An implicit COMMIT is issued before this procedure is executed. A session that is the first to signal an alert can also wait for the alert in a subsequent transaction. In this case, remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Syntax

```
DBMS_ALERT.WAITONE (
   name      IN   VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 2–8 WAITONE Procedure Parameters

Parameter	Description
name	Name of the alert to wait for.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITONE, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - time-out occurred
timeout	Maximum time to wait for an alert. If the named alert does not occurs before timeout seconds, this returns a status of 1.

Example

Suppose you want to graph average salaries by department, for all employees. Your application needs to know whenever EMP is changed. Your application would look similar to this code:

```
DBMS_ALERT.REGISTER('emp_table_alert');
    <<readagain>>:
   /* ... read the emp table and graph it */
      DBMS_ALERT.WAITONE('emp_table_alert', :message, :status);
      if status = 0 then goto <<readagain>>; else
      /* ... error condition */
```

The EMP table would have a trigger similar to this:

```
CREATE TRIGGER emptrig AFTER INSERT OR UPDATE OR DELETE ON emp
    BEGIN
      DBMS_ALERT.SIGNAL('emp_table_alert', 'message_text');
   END;
```

When the application is no longer interested in the alert, it makes this request:

```
DBMS_ALERT.REMOVE('emp_table_alert');
```

This reduces the amount of work required by the alert signaller. If a session exits (or dies) while registered alerts exist, the alerts are eventually cleaned up by future users of this package.

The preceding example guarantees that the application always sees the latest data, although it may not see every intermediate value.

3

DBMS_APPLICATION_INFO

Application developers can use the DBMS_APPLICATION_INFO package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules and debugging.

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the V$SESSION and V$SQLAREA views.

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather your own statistics based on module, you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the SYS version of the package. The public synonym for DBMS_APPLICATION_INFO can then be changed to point to the DBA's version of the package.

This chapter discusses the following topics:

Note: The public synonym for DBMS_APPLICATION_INFO is not dropped before creation so that you can redirect the public synonym to point to your own package.

Privileges

No further privileges are required. The DBMSUTIL.SQL script is already run by catproc.

Summary of DBMS_APPLICATION_INFO Subprograms

Table 3–1 DBMS_APPLICATION_INFO Package Subprograms

Subprogram	Description
SET_MODULE Procedure on page 3-2	Sets the name of the module that is currently running to a new module.
SET_ACTION Procedure on page 3-3	Sets the name of the current action within the current module.
READ_MODULE Procedure on page 3-4	Reads the values of the module and action fields of the current session.
SET_CLIENT_INFO Procedure on page 3-5	Sets the client info field of the session.
READ_CLIENT_INFO Procedure on page 3-6	Reads the value of the client_info field of the current session.
SET_SESSION_LONGOPS Procedure on page 3-6	Sets a row in the V$SESSION_LONGOP table.

SET_MODULE Procedure

This procedure sets the name of the current application or module. The module name should be the name of the procedure (if using stored procedures), or the name of the application. The action name should describe the action performed.

Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
   module_name IN VARCHAR2,
   action_name IN VARCHAR2);
```

Parameters

Table 3–2 SET_MODULE Procedure Parameters

Parameter	Description
module_name	Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.
action_name	Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.

Example

```
CREATE or replace PROCEDURE add_employee(
  name VARCHAR2,
  salary NUMBER,
  manager NUMBER,
  title VARCHAR2,
  commission NUMBER,
  department NUMBER) AS
BEGIN
  DBMS_APPLICATION_INFO.SET_MODULE(
    module_name => 'add_employee',
    action_name => 'insert into emp');
  INSERT INTO emp
    (ename, empno, sal, mgr, job, hiredate, comm, deptno)
    VALUES (name, emp_seq.nextval, salary, manager, title, SYSDATE,
            commission, department);
  DBMS_APPLICATION_INFO.SET_MODULE(null,null);
END;
```

SET_ACTION Procedure

This procedure sets the name of the current action within the current module. The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Syntax

```
DBMS_APPLICATION_INFO.SET_ACTION (
   action_name IN VARCHAR2);
```

Parameters

Table 3–3 SET_ACTION Procedure Parameters

Parameter	Description
action_name	The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.

Usage Notes

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, subsequent transactions may be logged with the previous transaction's name.

Example

The following is an example of a transaction that uses the registration procedure:

```
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN

-- balance transfer transaction

   DBMS_APPLICATION_INFO.SET_ACTION(
      action_name => 'transfer from chk to sav');
   UPDATE chk SET bal = bal + :amt
      WHERE acct# = :acct;
   UPDATE sav SET bal = bal - :amt
      WHERE acct# = :acct;
   COMMIT;
   DBMS_APPLICATION_INFO.SET_ACTION(null);

END;
```

READ_MODULE Procedure

This procedure reads the values of the module and action fields of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_MODULE (
   module_name OUT VARCHAR2,
```

```
   action_name OUT VARCHAR2);
```

Parameters

Table 3–4 READ_MODULE Procedure Parameters

Parameter	Description
module_name	Last value that the module name was set to by calling SET_MODULE.
action_name	Last value that the action name was set to by calling SET_ACTION or SET_MODULE.

Usage Notes

Module and action names for a registered application can be retrieved by querying V$SQLAREA or by calling the READ_MODULE procedure. Client information can be retrieved by querying the V$SESSION view, or by calling the READ_CLIENT_INFO procedure.

Example

The following sample query illustrates the use of the MODULE and ACTION column of the V$SQLAREA.

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';

SQL_TEXT DISK_READS MODULE ACTION
------------------ ---------- ----------------- ---------------
INSERT INTO emp 1 add_employee insert into emp
(ename, empno, sal, mgr, job, hiredate, comm, deptno)
VALUES
(name, next.emp_seq, manager, title, SYSDATE, commission, department)

1 row selected.
```

SET_CLIENT_INFO Procedure

This procedure supplies additional information about the client application.

Syntax

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (
```

```
   client_info IN VARCHAR2);
```

Parameters

Table 3–5 SET_CLIENT_INFO Procedure Parameters

Parameter	Description
client_info	Supplies any additional information about the client application. This information is stored in the V$SESSIONS view. Information exceeding 64 bytes is truncated.

Note: CLIENT_INFO is readable and writable by any user. For storing secured application attributes, you can use the application context feature.

READ_CLIENT_INFO Procedure

This procedure reads the value of the client_info field of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO (
   client_info OUT VARCHAR2);
```

Parameters

Table 3–6 READ_CLIENT_INFO Procedure Parameters

Parameter	Description
client_info	Last client information value supplied to the SET_CLIENT_INFO procedure.

SET_SESSION_LONGOPS Procedure

This procedure sets a row in the V$SESSION_LONGOPS view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.

Applications may use the set_session_longops procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the V$SESSION_LONGOPS view.

Syntax

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (
   rindex      IN OUT BINARY_INTEGER,
   slno        IN OUT BINARY_INTEGER,
   op_name     IN     VARCHAR2       DEFAULT NULL,
   target      IN     BINARY_INTEGER DEFAULT 0,
   context     IN     BINARY_INTEGER DEFAULT 0,
   sofar       IN     NUMBER         DEFAULT 0,
   totalwork   IN     NUMBER         DEFAULT 0,
   target_desc IN     VARCHAR2       DEFAULT 'unknown target',
   units       IN     VARCHAR2       DEFAULT NULL)

set_session_longops_nohint constant BINARY_INTEGER := -1;
```

Pragmas

```
pragma TIMESTAMP('1998-03-12:12:00:00');
```

Parameters

Table 3–7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
rindex	A token which represents the v$session_longops row to update. Set this to set_session_longops_nohint to start a new row. Use the returned value from the prior call to reuse a row.
slno	Saves information across calls to set_session_longops: It is for internal use and should not be modified by the caller.
op_name	Specifies the name of the long running task. It appears as the OPNAME column of v$session_longops. The maximum length is 64 bytes.
target	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of v$session_longops.
context	Any number the client wants to store. It appears in the CONTEXT column of v$session_longops.

Table 3–7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
sofar	Any number the client wants to store. It appears in the SOFAR column of v$session_longops. This is typically the amount of work which has been done so far.
totalwork	Any number the client wants to store. It appears in the TOTALWORK column of v$session_longops. This is typically an estimate of the total amount of work needed to be done in this long running operation.
target_desc	Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of v$session_longops. The maximum length is 32 bytes.
units	Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes.

Example

This example performs a task on 10 objects in a loop. As the example completes each object, Oracle updates V$SESSION_LONGOPS on the procedure's progress.

```
DECLARE
        rindex    BINARY_INTEGER;
        slno      BINARY_INTEGER;
        totalwork number;
        sofar     number;
        obj       BINARY_INTEGER;

      BEGIN
        rindex := dbms_application_info.set_session_longops_nohint;
        sofar := 0;
        totalwork := 10;

        WHILE sofar < 10 LOOP
          -- update obj based on sofar
          -- perform task on object target

          sofar := sofar + 1;
          dbms_application_info.set_session_longops(rindex, slno,
            "Operation X", obj, 0, sofar, totalwork, "table", "tables");
        END LOOP;
      END;
```

4

DBMS_APPLY_ADM

The DBMS_APPLY_ADM package provides administrative procedures to start, stop, and configure an apply process.

This chapter contains the following topic:

- Summary of DBMS_APPLY_ADM Subprograms

> **See Also:** *Oracle9i Streams* for more information about the apply process

Summary of DBMS_APPLY_ADM Subprograms

Table 4–1 DBMS_APPLY_ADM Subprograms **(Page 1 of 2)**

Subprogram	Description
"ALTER_APPLY Procedure" on page 4-4	Alters an apply process
"CREATE_APPLY Procedure" on page 4-9	Creates an apply process
"DELETE_ALL_ERRORS Procedure" on page 4-13	Deletes all the error transactions for the specified apply process from the error queue
"DELETE_ERROR Procedure" on page 4-14	Deletes the specified error transaction from the error queue
"DROP_APPLY Procedure" on page 4-14	Drops an apply process
"EXECUTE_ALL_ERRORS Procedure" on page 4-15	Reexecutes the error queue transactions for the specified apply process.
"EXECUTE_ERROR Procedure" on page 4-16	Reexecutes the specified error queue transaction
"GET_ERROR_MESSAGE Function" on page 4-17	Returns the message payload from the error queue for the specified message number and transaction identifier
"SET_DML_HANDLER Procedure" on page 4-18	Alters operation options for a specified object with a specified apply process
"SET_GLOBAL_INSTANTIATION_SCN Procedure" on page 4-23	Records the specified instantiation SCN for the specified source database
"SET_KEY_COLUMNS Procedure" on page 4-26	Records the set of columns to be used as the substitute primary key for local apply purposes and removes existing substitute primary key columns for the specified object if they exist
"SET_PARAMETER Procedure" on page 4-28	Sets an apply parameter to the specified value
"SET_SCHEMA_INSTANTIATION_SCN Procedure" on page 4-32	Records the specified instantiation SCN for the specified schema in the specified source database
"SET_TABLE_INSTANTIATION_SCN Procedure" on page 4-35	Records the specified instantiation SCN for the specified table in the specified source database

Table 4–1 DBMS_APPLY_ADM Subprograms **(Page 2 of 2)**

Subprogram	Description
"SET_UPDATE_CONFLICT_HANDLER Procedure" on page 4-37	Adds, updates, or drops an update conflict handler for the specified object
"START_APPLY Procedure" on page 4-41	Directs the apply process to start applying events
"STOP_APPLY Procedure" on page 4-42	Stops the apply process from applying any events and rolls back any unfinished transactions being applied

Note: All procedures and functions commit unless specified otherwise.

ALTER_APPLY Procedure

Alters an apply process.

Syntax

```
DBMS_APPLY_ADM.ALTER_APPLY(
   apply_name              IN  VARCHAR2,
   rule_set_name           IN  VARCHAR2  DEFAULT NULL,
   remove_rule_set         IN  BOOLEAN   DEFAULT false,
   message_handler         IN  VARCHAR2  DEFAULT NULL
   remove_message_handler  IN  BOOLEAN   DEFAULT false,
   ddl_handler             IN  VARCHAR2  DEFAULT NULL,
   remove_ddl_handler      IN  BOOLEAN   DEFAULT false,
   apply_user              IN  VARCHAR2  DEFAULT NULL,
   apply_tag               IN  RAW       DEFAULT NULL,
   remove_apply_tag        IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 4–2 ALTER_APPLY Procedure Parameters **(Page 1 of 5)**

Parameter	Description
apply_name	The name of the apply process being altered. You must specify an existing apply process name.
rule_set_name	The name of the rule set that contains the apply rules for this apply process. If you want to use a rule set for the apply process, then you must specify an existing rule set in the form [*schema_name.*]*rule_set_name*. For example, to specify a rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_RULE_ADM package. If you specify NULL, then the apply process applies all LCRs and user messages in its queue.

Table 4–2 ALTER_APPLY Procedure Parameters **(Page 2 of 5)**

Parameter	Description
remove_rule_set	If true, then removes the rule set for the specified apply process. If false, then retains any rule set for the specified apply process. If the rule_set_name parameter is non-NULL, then this parameter should be set to false.
message_handler	A user-defined procedure that processes non-LCR messages in the queue for the apply process. You must specify an existing procedure in one of the following forms: ▪ [*schema_name.*]*procedure_name* ▪ [*schema_name.*]*package_name.procedure_name* If the procedure is in a package, then the *package_name* must be specified. For example, to specify a procedure in the apply_pkg package in the hr schema named process_msgs, enter hr.apply_pkg.process_msgs. An error is returned if the specified procedure does not exist. If the schema is not specified, then the user who invokes the ALTER_APPLY procedure is the default. This user must have EXECUTE privilege on a specified message handler.
remove_message_handler	If true, then removes the message handler for the specified apply process. If false, then retains any message handler for the specified apply process. If the message_handler parameter is non-NULL, then this parameter should be set to false.

Table 4–2 ALTER_APPLY Procedure Parameters **(Page 3 of 5)**

Parameter	Description
ddl_handler	A user-defined procedure that processes DDL LCRs in the queue for the apply process. You must specify an existing procedure in the form [*schema_name*.]*procedure_name*. For example, to specify a procedure in the hr schema named process_ddls, enter hr.process_ddls. An error is returned if the specified procedure does not exist. If the schema is not specified, then the user who invokes the ALTER_APPLY procedure is the default. This user must have EXECUTE privilege on a specified DDL handler. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically.
remove_ddl_handler	If true, then removes the DDL handler for the specified apply process. If false, then retains any DDL handler for the specified apply process. If the ddl_handler parameter is non-NULL, then this parameter should be set to false.

Table 4–2 ALTER_APPLY Procedure Parameters **(Page 4 of 5)**

Parameter	Description
apply_user	The user who applies all DML and DDL changes and who runs user-defined apply handlers. If NULL, then the apply user is not changed. The specified user must have the necessary privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. The specified user must also have dequeue privileges on the queue used by the apply process and privileges to execute the rule set and transformation functions used by the apply process. These privileges must be granted directly to the apply user; they cannot be granted through roles. By default, this parameter is set to the user who created the apply process by running either the CREATE_APPLY procedure in this package or one of the following procedures in the DBMS_STREAMS_ADM package with the streams_type parameter set to apply: ▪ ADD_GLOBAL_RULES ▪ ADD_SCHEMA_RULES ▪ ADD_TABLE_RULES ▪ ADD_SUBSET_RULES **Note:** If the specified user is dropped using DROP USER ... CASCADE, then the apply_user for the apply process is set to NULL automatically. You must specify an apply user before the apply process can run.
apply_tag	A binary tag that is added to redo entries generated by the specified apply process. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply process is running will capture changes made by the apply process. If so, then the captured changes will include the tag specified by this parameter. If NULL, the default, then the apply tag for the apply process is not changed. The following is an example of a tag with a hexadecimal value of 17: HEXTORAW('17') **See Also:** *Oracle9i Streams* for more information about tags

Table 4–2 ALTER_APPLY Procedure Parameters **(Page 5 of 5)**

Parameter	Description
remove_apply_tag	If true, then sets the apply tag for the specified apply process to NULL, and the apply process generated redo entries with NULL tags. If false, then retains any apply tag for the specified apply process. If the apply_tag parameter is non-NULL, then this parameter should be set to false.

Usage Notes

An apply process is stopped and restarted automatically when you change the value of one or more of the following ALTER_APPLY procedure parameters:

- message_handler
- ddl_handler
- apply_user
- apply_tag

CREATE_APPLY Procedure

Creates an apply process.

Syntax

```
DBMS_APPLY_ADM.CREATE_APPLY(
   queue_name           IN  VARCHAR2,
   apply_name           IN  VARCHAR2,
   rule_set_name        IN  VARCHAR2  DEFAULT NULL,
   message_handler      IN  VARCHAR2  DEFAULT NULL,
   ddl_handler          IN  VARCHAR2  DEFAULT NULL,
   apply_user           IN  VARCHAR2  DEFAULT NULL,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL,
   apply_tag            IN  RAW       DEFAULT '00',
   apply_captured       IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 4–3 CREATE_APPLY Procedure Parameters **(Page 1 of 4)**

Parameter	Description
queue_name	The name of the queue from which the apply process dequeues LCRs and user messages. You must specify an existing queue in the form [*schema_name*.]*queue_name*. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default. **Note:** The queue_name setting cannot be altered after the apply process is created.
apply_name	The name of the apply process being created. A NULL specification is not allowed. **Note:** The apply_name setting cannot be altered after the apply process is created.

Table 4–3 CREATE_APPLY Procedure Parameters **(Page 2 of 4)**

Parameter	Description
rule_set_name	The name of the rule set that contains the apply rules for this apply process. If you want to use a rule set for the apply process, then you must specify an existing rule set in the form [*schema_name*.]*rule_set_name*. For example, to specify a rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_RULE_ADM package. If you specify NULL, then the apply process applies all LCRs and user messages in its queue.
message_handler	A user-defined procedure that processes non-LCR messages in the queue for the apply process. You must specify an existing procedure in one of the following forms: ▪ [*schema_name*.]*procedure_name* ▪ [*schema_name*.]*package_name.procedure_name* If the procedure is in a package, then the *package_name* must be specified. For example, to specify a procedure in the apply_pkg package in the hr schema named process_msgs, enter hr.apply_pkg.process_msgs. An error is returned if the specified procedure does not exist. If the schema is not specified, then the user who invokes the CREATE_APPLY procedure is the default. This user must have EXECUTE privilege on a specified message handler. See "Usage Notes" on page 4-13 for more information about a message handler procedure.

Table 4–3 CREATE_APPLY Procedure Parameters **(Page 3 of 4)**

Parameter	Description
ddl_handler	A user-defined procedure that processes DDL LCRs in the queue for the apply process. You must specify an existing procedure in one of the following forms: ▪ [*schema_name*.]*procedure_name* ▪ [*schema_name*.]*package_name.procedure_name* If the procedure is in a package, then the *package_name* must be specified. For example, to specify a procedure in the apply_pkg package in the hr schema named process_ddls, enter hr.apply_pkg.process_ddls. An error is returned if the specified procedure does not exist. If the schema is not specified, then the user who invokes the CREATE_APPLY procedure is the default. This user must have EXECUTE privilege on a specified DDL handler. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically. See "Usage Notes" on page 4-13 for more information about a DDL handler procedure.
apply_user	The user who applies all DML and DDL changes and who runs user-defined apply handlers. If NULL, then the user who runs the CREATE_APPLY procedure is used. The user must have the necessary privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. The specified user must also have dequeue privileges on the queue used by the apply process and privileges to execute the rule set and transformation functions used by the apply process. These privileges must be granted directly to the apply user; they cannot be granted through roles. **Note:** If the specified user is dropped using DROP USER . . . CASCADE, then the apply_user setting for the apply process is set to NULL automatically. You must specify an apply user before the apply process can run. **See Also:** *Oracle9i Streams* for more information about the privileges required to apply changes

Table 4–3 CREATE_APPLY Procedure Parameters **(Page 4 of 4)**

Parameter	Description
apply_database_link	The database at which the apply process applies messages. This parameter is used by an apply process when applying changes from Oracle to non-Oracle systems, such as Sybase. Set this parameter to NULL to specify that the apply process applies messages at the local database. **Note:** The apply_database_link setting cannot be altered after the apply process is created.
apply_tag	A binary tag that is added to redo entries generated by the specified apply process. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply process is running will capture changes made by the apply process. If so, then the captured changes will include the tag specified by this parameter. By default, the tag for an apply process is the hexadecimal equivalent of '00' (double zero). The following is an example of a tag with a hexadecimal value of 17: HEXTORAW('17') If NULL, then the apply process generates redo entries with NULL tags. **See Also:** *Oracle9i Streams* for more information about tags
apply_captured	Either true or false. If true, then the apply process applies only the events in a queue that were captured by a Streams capture process. If false, then the apply process applies only the user-enqueued events in a queue. These events are user messages that were not captured by a Streams capture process. These messages may or may not contain a user-created LCR. To apply both captured and user-enqueued events in a queue, you must create at least two apply processes. **Note:** The apply_captured setting cannot be altered after the apply process is created. **See Also:** *Oracle9i Streams* for more information about captured and user-enqueued events

Usage Notes

The procedure specified in both the `message_handler` parameter and the `ddl_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name   IN  SYS.AnyData);
```

Here, *`handler_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. For the message handler, the parameter passed to the procedure is a `SYS.AnyData` encapsulation of a user message. For the DDL handler procedure, the parameter passed to the procedure is a `SYS.AnyData` encapsulation of a DDL LCR.

> **See Also:** Chapter 108, "Logical Change Record Types" for information DDL LCRs

DELETE_ALL_ERRORS Procedure

Deletes all the error transactions for the specified apply process from the error queue.

Syntax

```
DBMS_APPLY_ADM.DELETE_ALL_ERRORS(
   apply_name     IN VARCHAR2 DEFAULT NULL);
```

Parameter

Table 4–4 DELETE_ALL_ERRORS Procedure Parameter

Parameter	Description
`apply_name`	The name of the apply process that raised the errors while processing the transactions. If `NULL`, then all error transactions for all apply processes are deleted.

DELETE_ERROR Procedure

Deletes the specified error transaction from the error queue.

Syntax

```
DBMS_APPLY_ADM.DELETE_ERROR(
   local_transaction_id    IN VARCHAR2);
```

Parameter

Table 4–5 DELETE_ERROR Procedure Parameter

Parameter	Description
local_transaction_id	The identification number of the error transaction to delete. If the specified transaction does not exist in the error queue, then an error is raised.

DROP_APPLY Procedure

Drops an apply process.

Syntax

```
DBMS_APPLY_ADM.DROP_APPLY(
   apply_name          IN VARCHAR2);
```

Parameter

Table 4–6 DROP_APPLY Procedure Parameter

Parameter	Description
apply_name	The name of the apply process being dropped. You must specify an existing apply process name.

EXECUTE_ALL_ERRORS Procedure

Reexecutes the error queue transactions for the specified apply process.

The transactions are reexecuted in commit SCN order. Error reexecution stops if an error is raised.

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
   apply_name           IN VARCHAR2  DEFAULT NULL
   execute_as_user      IN BOOLEAN   DEFAULT false);
```

Parameters

Table 4–7 EXECUTE_ALL_ERRORS Procedure Parameters

Parameter	Description
apply_name	The name of the apply process that raised the errors while processing the transactions. If NULL, then all error transactions for all apply processes are reexecuted.
execute_as_user	If TRUE, then reexecutes the transactions in the security context of the current user. If FALSE, then reexecutes each transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each transaction in the error queue. The user who executes the transactions must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply process.

EXECUTE_ERROR Procedure

Reexecutes the specified error queue transaction.

Syntax

```
DBMS_APPLY_ADM.EXECUTE_ERROR(
   local_transaction_id     IN VARCHAR2,
   execute_as_user          IN BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 4–8 EXECUTE_ERROR Procedure Parameters

Parameter	Description
local_transaction_id	The identification number of the error transaction to execute. If the specified transaction does not exist in the error queue, then an error is raised.
execute_as_user	If TRUE, then reexecutes the transaction in the security context of the current user. If FALSE, then reexecutes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each transaction in the error queue. The user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply process.

GET_ERROR_MESSAGE Function

Returns the message payload from the error queue for the specified message number and transaction identifier.

Syntax

```
DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
   message_number          IN NUMBER,
   local_transaction_id    IN VARCHAR2)
RETURN Sys.Anydata;
```

Parameters

Table 4–9 GET_ERROR_MESSAGE Function Parameters

Parameter	Description
`message_number`	The identification number of the message. Query the DBA_APPLY_ERROR data dictionary view to view the message number of each apply error.
`local_transaction_id`	Identifier of the error transaction for which to return a message

SET_DML_HANDLER Procedure

Sets a user procedure as a DML handler for a specified operation on a specified object. The user procedure alters the apply behavior for the specified operation on the specified object. Run this procedure at the destination database. The SET_DML_HANDLER procedure provides a way for users to apply logical change records containing DML changes (row LCRs) using a customized apply.

If the error_handler parameter is set to true, then it specifies that the user procedure is an error handler. An error handler is invoked only when a row LCR raises an apply process error. Such an error may result from a data conflict if no conflict handler is specified or if the update conflict handler cannot resolve the conflict. If the error_handler parameter is set to false, then the user procedure is a DML handler, not an error handler, and a DML handler is always run instead of performing the specified operation on the specified object.

This procedure either sets a DML handler or an error handler for a particular operation on an object. It cannot set both a DML handler and an error handler for the same object and operation.

At the source database, you must specify an unconditional supplemental log group for the columns needed by a DML or error handler.

Note: Currently, setting an error handler for an apply process that is applying changes to a non-Oracle database is not supported.

Syntax

```
DBMS_APPLY_ADM.SET_DML_HANDLER(
   object_name          IN  VARCHAR2,
   object_type          IN  VARCHAR2,
   operation_name       IN  VARCHAR2,
   error_handler        IN  BOOLEAN  DEFAULT false,
   user_procedure       IN  VARCHAR2,
   apply_database_link  IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–10 SET_DML_HANDLER Procedure Parameters **(Page 1 of 2)**

Parameter	Description
object_name	The name of the source object specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
object_type	The type of the source object. Currently, TABLE is the only possible source object type.
operation_name	The name of the operation, which can be specified as: ▪ INSERT ▪ UPDATE ▪ DELETE ▪ LOB_UPDATE For example, suppose you run this procedure twice for the hr.employees table. In one call, you set operation_name to UPDATE and user_procedure to employees_update. In another call, you set operation_name to INSERT and user_procedure to employees_insert. Both times, you set error_handler to false. In this case, the employees_update procedure is run for UPDATE operations on the hr.employees table, and the employees_insert procedure is run for INSERT operations on the hr.employees table.
error_handler	If true, then the specified user procedure is run when a row LCR involving the specified operation on the specified object raises an apply process error. The user procedure may try to resolve possible error conditions, or it may simply notify administrators of the error or log the error. If false, then the handler being set is run for all row LCRs involving the specified operation on the specified object. **Note:** Currently, error handlers are not supported when applying changes to a non-Oracle database.

Table 4–10 SET_DML_HANDLER Procedure Parameters **(Page 2 of 2)**

Parameter	Description
user_procedure	A user-defined procedure that is invoked during apply for the specified operation on the specified object. If the procedure is a DML handler, then it is invoked instead of the default apply performed by Oracle. If the procedure is an error handler, then it is invoked when the apply process encounters an error.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

Usage Notes

The SET_DML_HANDLER procedure can be used to set either a general DML handler or an error handler for row LCRs that perform a specified operation on a specified object. The following sections describe the signature of a general DML handler procedure and the signature of an error handler procedure.

In either case, you must specify the full procedure name for the user_procedure parameter in one of the following forms:

- [*schema_name*.]p*ackage_name.procedure_name*
- [*schema_name*.]*procedure_name*

If the procedure is in a package, then the *package_name* must be specified. If the schema is not specified, then the user who invokes the SET_DML_HANDLER procedure is the default. This user must have EXECUTE privilege on the specified procedure.

For example, suppose the procedure_name has the following properties:

- hr is the *schema_name*.
- apply_pkg is the *package_name*.
- employees_default is the *procedure_name*.

In this case, specify the following:

```
hr.apply_pkg.employees_default
```

The following restrictions apply to the user procedure:

- Do not execute `COMMIT` or `ROLLBACK` statements. Doing so may endanger the consistency of the transaction that contains the LCR.
- If you are manipulating a row using the `EXECUTE` member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.
- If the command type is `UPDATE` or `DELETE`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR must include the entire key in the list of old values. The key is the primary key, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure.
- If the command type is `INSERT`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure.

Signature of a General DML Handler Procedure

The procedure specified in the `user_procedure` parameter must have the following signature:

```
PROCEDURE user_procedure (
   parameter_name   IN  SYS.AnyData);
```

Here, *`user_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a `SYS.AnyData` encapsulation of a row LCR.

> **See Also:** Chapter 108, "Logical Change Record Types" for more information about LCRs

Signature of an Error Handler Procedure

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
     message             IN SYS.AnyData,
     error_stack_depth   IN NUMBER,
     error_numbers       IN DBMS_UTILITY.NUMBER_ARRAY,
     error_messages      IN emsg_array);
```

Note:

- Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.
- The `emsg_array` parameter must be a user-defined array that is a table of type VARCHAR2 with at least 76 characters.

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error and returns control to the apply process.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.

If you want to retry the DML operation, then have the error handler procedure run the EXECUTE member procedure for the LCR.

SET_GLOBAL_INSTANTIATION_SCN Procedure

Records the specified instantiation SCN for the specified source database. This procedure overwrites any existing instantiation SCN for the database.

This procedure gives you precise control over which DDL LCRs for a database are ignored and which DDL LCRs are applied by an apply process. If the commit SCN of a DDL LCR for a database object from a source database is less than or equal to the instantiation SCN for that database at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

The instantiation SCN specified by this procedure is used for a DDL LCR only if the DDL LCR does not have `object_owner`, `base_table_owner`, and `base_table_name` specified. For example, the instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `CREATE USER`.

Attention: If you run the `SET_GLOBAL_INSTANTIATION_SCN` for a database, then you should run `SET_SCHEMA_INSTANTIATION_SCN` for all of the existing schemas in the database and `SET_TABLE_INSTANTIATION_SCN` for all of the existing tables in the database. If you add new schemas and tables to the database in the future, then you need not run these procedures for the new schemas and tables.

Note:

- This procedure sets the instantiation SCN only for DDL LCRs. To set the instantiation SCN for row LCRs, which record the results of DML changes, use `SET_TABLE_INSTANTIATION_SCN`.
- The instantiation SCN set by the `SET_SCHEMA_INSTANTIATION_SCN` procedure is used for DDL LCRs that have `object_owner` specified.
- The instantiation SCN set by the `SET_TABLE_INSTANTIATION_SCN` procedure is used for DDL LCRs that have both `base_table_owner` and `base_table_name` specified, except for DDL LCRs with a `command_type` of `CREATE TABLE`.
- The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- "SET_SCHEMA_INSTANTIATION_SCN Procedure" on page 4-32
- "SET_TABLE_INSTANTIATION_SCN Procedure" on page 4-35
- "LCR$_DDL_RECORD Type" on page 108-3 for more information about DDL LCRs
- *Oracle9i Streams*

Syntax

```
DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
  source_database_name   IN  VARCHAR2,
  instantiation_scn      IN  NUMBER,
  apply_database_link    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–11 SET_GLOBAL_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_database_name	The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the local domain is appended to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then DBS1.NET is specified automatically.
instantiation_scn	The instantiation SCN number. Specify NULL to remove the instantiation SCN metadata for the source database from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.

SET_KEY_COLUMNS Procedure

Records the set of columns to be used as the substitute primary key for apply purposes and removes existing substitute primary key columns for the specified object if they exist. Unlike true primary keys, these columns may contain NULLs.

When not empty, this set of columns takes precedence over any primary key for the specified object. Do not specify substitute key columns if the object already has primary key columns and you want to use those primary key columns as the key.

Run this procedure at the destination database. At the source database, you must specify an unconditional supplemental log group for the substitute key columns.

Note:

- Oracle Corporation recommends that each column you specify as a substitute key column be a NOT NULL column. You should also create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because Oracle can locate the relevant row more efficiently.
- You should not permit applications to update the primary key or substitute key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of the data.

Note: This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Syntax

```
DBMS_APPLY_ADM.SET_KEY_COLUMNS(
   object_name            IN   VARCHAR2,
   { column_list          IN   VARCHAR2, |
     column_table         IN   DBMS_UTILITY.NAME_ARRAY, }
   apply_database_link    IN   VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–12 SET_KEY_COLUMNS Procedure Parameters

Parameter	Description
`object_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default. If the apply process is applying changes to a non-Oracle database in a heterogeneous environment, then the object name is not verified.
`column_list`	A comma-delimited list of the columns in the table that you want to use as the substitute primary key, with no spaces between the column names. If the `column_list` parameter is empty or `NULL`, then the current set of key columns is removed.
`column_table`	A PL/SQL index-by table of type `DBMS_UTILITY.NAME_ARRAY` of the columns in the table that you want to use as the substitute primary key. The index for `column_table` must be 1-based, increasing, dense, and terminated by a `NULL`. If the `column_table` parameter is empty or `NULL`, then the current set of key columns is removed.
`apply_database_link`	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

SET_PARAMETER Procedure

Sets an apply parameter to the specified value.

When you alter a parameter value, a short amount of time may pass before the new value for the parameter takes effect.

Syntax

```
DBMS_APPLY_ADM.SET_PARAMETER (
   apply_name      IN VARCHAR2,
   parameter       IN VARCHAR2,
   value           IN VARCHAR2);
```

Parameters

Table 4–13 SET_PARAMETER Procedure Parameters

Parameter	Description
`apply_name`	The apply process name
`parameter`	The name of the parameter you are setting. See "Apply Process Parameters" on page 4-29 for a list of these parameters.
`value`	The value to which the parameter is set

Apply Process Parameters

The following table lists the parameters for the apply process.

Table 4–14 Apply Process Parameters **(Page 1 of 3)**

Parameter Name	Possible Values	Default	Description
commit_serialization	full or none	full	The order in which applied transactions are committed. If full, then the apply process commits applied transactions in the order in which they were committed at the source database. If none, then the apply process may commit transactions may commit in any order. Performance is best if you specify none. Regardless of the specification, applied transactions may execute in parallel subject to data dependencies and constraint dependencies. Logical standby environments typically specify full.
disable_on_error	Y or N	Y	If Y, then the apply process is disabled on the first unresolved error, even if the error is not fatal. If N, then the apply process continues regardless of unresolved errors.
disable_on_limit	Y or N	N	If Y, then the apply process is disabled if the apply process terminates because it reached a value specified by the time_limit parameter or transaction_limit parameter. If N, then the apply process is restarted immediately after stopping because it reached a limit.
maximum_scn	A valid SCN or infinite	infinite	The apply process is disabled before applying a transaction with a commit SCN greater than or equal to the value specified. If infinite, then the apply process runs regardless of the SCN value.

Table 4–14 Apply Process Parameters **(Page 2 of 3)**

Parameter Name	Possible Values	Default	Description
parallelism	A positive integer	1	The number of transactions that may be concurrently applied **Note:** ▪ When you change the value of this parameter, the apply process is stopped and restarted automatically. This may take some time depending on the size of the transactions currently being applied. ▪ Setting the parallelism parameter to a number higher than the number of available parallel execution servers may disable the apply process. Make sure the PROCESSES and PARALLEL_MAX_SERVERS initialization parameters are set appropriately when you set the parallelism apply process parameter.
startup_seconds	0, a positive integer, or infinite	0	The maximum number of seconds to wait for another instantiation of the same apply process to finish. If the other instantiation of the same apply process does not finish within this time, then the apply process does not start. If infinite, then an apply process does not start until another instantiation of the same apply process finishes.
time_limit	A positive integer or infinite	infinite	The apply process stops as soon as possible after the specified number of seconds since it started. If infinite, then the apply process continues to run until it is stopped explicitly.
trace_level	0 or a positive integer	0	Set this parameter only under the guidance of Oracle Support Services.

Table 4–14 Apply Process Parameters **(Page 3 of 3)**

Parameter Name	Possible Values	Default	Description
`transaction_limit`	A positive integer or `infinite`	`infinite`	The apply process stops after applying the specified number of transactions. If `infinite`, then the apply process continues to run regardless of the number of transactions applied.
`write_alert_log`	`Y` or `N`	`Y`	If `Y`, then the apply process writes a message to the alert log on exit. If `N`, then the apply process does not write a message to the alert log on exit. The message specifies the reason why the apply process stopped.

Note:

- For all parameters that are interpreted as positive integers, the maximum possible value is `4,294,967,295`. Where applicable, specify `infinite` for larger values.
- For parameters that require an SCN setting, any valid SCN value can be specified.

SET_SCHEMA_INSTANTIATION_SCN Procedure

Records the specified instantiation SCN for the specified schema in the specified source database. This procedure overwrites any existing instantiation SCN for the particular schema.

This procedure gives you precise control over which DDL LCRs for a schema are ignored and which DDL LCRs are applied by an apply process. If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

The instantiation SCN specified by this procedure is used on the following types of DDL LCRs:

- DDL LCRs with a `command_type` of `CREATE TABLE`
- DDL LCRs with a non-`NULL` `object_owner` specified and no `base_table_owner` nor `base_table_name` specified.

For example, the instantiation SCN set by this procedure is used for a DDL LCR with a `command_type` of `CREATE TABLE` and `ALTER USER`.

The instantiation SCN specified by this procedure is not used for DDL LCRs with a `command_type` of `CREATE USER`.

Attention: If you run the `SET_SCHEMA_INSTANTIATION_SCN` for a schema, then you should run `SET_TABLE_INSTANTIATION_SCN` for all of the existing tables in the schema. If you add new tables to the schema in the future, then you need not run `SET_TABLE_INSTANTIATION_SCN` for these tables.

Note:

- This procedure sets the instantiation SCN only for DDL LCRs. To set the instantiation SCN for row LCRs, which record the results of DML changes, use `SET_TABLE_INSTANTIATION_SCN`.
- The instantiation SCN set by the `SET_TABLE_INSTANTIATION_SCN` procedure is used for DDL LCRs that have both `base_table_owner` and `base_table_name` specified, except for DDL LCRs with a `command_type` of `CREATE TABLE`.
- The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- "SET_GLOBAL_INSTANTIATION_SCN Procedure" on page 4-23
- "SET_TABLE_INSTANTIATION_SCN Procedure" on page 4-35
- "LCR$_DDL_RECORD Type" on page 108-3 for more information about DDL LCRs
- *Oracle9i Streams*

Syntax

```
DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
  source_schema_name     IN  VARCHAR2,
  source_database_name   IN  VARCHAR2,
  instantiation_scn      IN  NUMBER,
  apply_database_link    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–15 SET_SCHEMA_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_schema_name	The name of the source schema. For example, hr.
source_database_name	The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the local is appended to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then DBS1.NET is specified automatically.
instantiation_scn	The instantiation SCN number. Specify NULL to remove the instantiation SCN metadata for the source schema from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.

SET_TABLE_INSTANTIATION_SCN Procedure

Records the specified instantiation SCN for the specified table in the specified source database. This procedure overwrites any existing instantiation SCN for the particular table.

This procedure gives you precise control over which LCRs for a table are ignored and which LCRs are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at some destination database, then the apply process at the destination database disregards the LCR. Otherwise, the apply process applies the LCR.

The instantiation SCN specified by this procedure is used on the following types of LCRs:

- Row LCRs for the table
- DDL LCRs that have a non-`NULL` `base_table_owner` and `base_table_name` specified, except for DDL LCRs with a `command_type` of `CREATE TABLE`

For example, the instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `ALTER TABLE` or `CREATE TRIGGER`.

Note: The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:

- "SET_GLOBAL_INSTANTIATION_SCN Procedure" on page 4-23
- "SET_SCHEMA_INSTANTIATION_SCN Procedure" on page 4-32
- "LCR$_ROW_RECORD Type" on page 108-15 for more information about row LCRs
- "LCR$_DDL_RECORD Type" on page 108-3 for more information about DDL LCRs
- *Oracle9i Streams*

Syntax

```
DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
  source_object_name     IN  VARCHAR2,
  source_database_name   IN  VARCHAR2,
  instantiation_scn      IN  NUMBER,
  apply_database_link    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–16 SET_TABLE_INSTANTIATION_SCN Procedure Parameters

Parameter	Description
source_object_name	The name of the source object specified as [*schema_name.*]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
source_database_name	The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the local domain name is appended to the database name automatically. For example, if you specify DBS1 and the global domain is .NET, then DBS1.NET is specified automatically.
instantiation_scn	The instantiation SCN number. Specify NULL to remove the instantiation SCN metadata for the source table from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply process is a non-Oracle database.

SET_UPDATE_CONFLICT_HANDLER Procedure

Adds, modifies, or removes an update conflict handler for the specified object.

If you want to modify an existing update conflict handler, then you specify the table and resolution column of an the existing update conflict handler. You can modify the prebuilt method or the column list.

If you want to remove an existing update conflict handler, then specify `NULL` for the prebuilt method and specify the table, column list, and resolution column of the existing update conflict handler.

If an update conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate update conflict handler to resolve the conflict
2. If no update conflict handler is specified or if the update conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply process, table, and operation to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

---

Note: Currently, setting an update conflict handler for an apply process that is applying to a non-Oracle database is not supported.

See Also: "Signature of an Error Handler Procedure" on page 4-22 for information about setting an error handler

Syntax

```
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
   object_name           IN  VARCHAR2,
   method_name           IN  VARCHAR2,
   resolution_column     IN  VARCHAR2,
   column_list           IN  DBMS_UTILITY.NAME_ARRAY,
   apply_database_link   IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 4–17 SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
object_name	The schema and name of the table, specified as [*schema_name.*]*object_name*, for which an update conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table employees owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.
method_name	Type of update conflict handler to create. You can specify one of the built-in handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained: ■ MAXIMUM: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database. ■ MINIMUM: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database. ■ OVERWRITE: Applies the column list from the source database, overwriting the column values at the destination database ■ DISCARD: Retains the column list from the destination database, discarding the column list from the source database If NULL, then removes any existing update conflict handler with the same object_name, resolution_column, and column_list. If non-NULL, then replaces any existing update conflict handler with the same object_name and resolution_column.

Table 4–17 SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

Parameter	Description
resolution_column	Name of the column used to uniquely identify an update conflict handler. For the MAXIMUM and MINIMUM prebuilt methods, the resolution column is also used to resolve the conflict. The resolution column must be one of the columns listed in the column_list parameter. NULL is not allowed for this parameter. For the OVERWRITE and DISCARD prebuilt methods, you can any column in the column list.
column_list	List of columns for which the conflict handler is called. If a conflict occurs for one or more of the columns in the list when an apply process tries to apply a row LCR, then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. **Note:** Conflict resolution does not support LOB columns. Therefore, you should not include LOB columns in the column_list parameter.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database. **Note:** Currently, conflict handlers are not supported when applying changes to a non-Oracle database.

Usage Notes

The following is an example for setting an update conflict handler for the `employees` table in the `hr` schema:

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name          =>  'hr.employees',
    method_name          =>  'MAXIMUM',
    resolution_column    =>  'salary',
    column_list          =>  cols);
END;
/
```

This example sets a conflict handler that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

START_APPLY Procedure

Directs the apply process to start applying events.

The start status is persistently recorded. Hence, if the status is START, then the apply process is started upon database instance startup. Each apply process is an Oracle background process and is prefixed by AP.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the start status of an apply process.

You can create the apply process using the following procedures:

- DBMS_APPLY_ADM.CREATE_APPLY
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES

See Also: Chapter 73, "DBMS_STREAMS_ADM"

Syntax

```
DBMS_APPLY_ADM.START_APPLY(
   apply_name  IN VARCHAR2);
```

Parameter

Table 4–18 START_APPLY Procedure Parameter

Parameter	Description
apply_name	The apply process name. A NULL setting is not allowed.

STOP_APPLY Procedure

Stops the apply process from applying events and rolls back any unfinished transactions being applied.

The stop status is persistently recorded. Hence, if the status is STOP, then the apply process is not started upon database instance startup.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the STOP status of an apply process.

Syntax

```
DBMS_APPLY_ADM.STOP_APPLY(
   apply_name   IN  VARCHAR2
   force        IN  BOOLEAN DEFAULT false);
```

Parameters

Table 4–19 STOP_APPLY Procedure Parameters

Parameter	Description
apply_name	The apply process name. A NULL setting is not allowed.
force	If true, then stops the apply process as soon as possible. If false, then stops the apply process after ensuring that there are no gaps in the set of applied transactions. The behavior of the apply process depends on the setting specified for the force parameter and the setting specified for the commit_serialization apply process parameter. See "Usage Notes" for more information.

Usage Notes

The following table describes apply process behavior for each setting of the force parameter in the STOP_APPLY procedure and the commit_serialization apply process parameter. In all cases, the apply process rolls back any unfinished transactions when it stops.

force	commit_serialization	Apply Process Behavior
true	full	The apply process stops immediately and does not apply any unfinished transactions.
true	none	When the apply process stops, some transactions that have been applied locally may have committed at the source database at a later point in time than some transactions that have not been applied locally.
false	full	The apply process stops after applying the next uncommitted transaction in the commit order, if any such transaction is in progress.
false	none	Before stopping, the apply process applies all of the transactions that have a commit time that is earlier than the applied transaction with the most recent commit time.

For example, assume that the commit_serialization apply process parameter is set to none and there are three transactions: transaction 1 has the earliest commit time, transaction 2 is committed after transaction 1, and transaction 3 has the latest commit time. Also assume that an apply process has applied transaction 1 and transaction 3 and is in the process of applying transaction 2 when the STOP_APPLY procedure is run. Given this scenario, if the force parameter is set to true, then transaction 2 is not applied, and the apply process stops (transaction 2 is rolled back). If, however, the force parameter is set to false, then transaction 2 is applied before the apply process stops.

A different scenario would result if the commit_serialization apply process parameter is set to full. For example, assume that the commit_serialization apply process parameter is set to full and there are three transactions: transaction A has the earliest commit time, transaction B is committed after transaction A, and transaction C has the latest commit time. In this case, the apply process has applied transaction A and is in the process of applying transactions B and C when the STOP_APPLY procedure is run. Given this scenario, if the force parameter is set to true, then transactions B and C are not applied, and the apply process stops (transactions B and C are rolled back). If, however, the force parameter is set to false, then transaction B is applied before the apply process stops, and transaction C is rolled back.

> **See Also:** "SET_PARAMETER Procedure" on page 4-28 for more information about the commit_serialization apply process parameter

5

DBMS_AQ

The DBMS_AQ package provides an interface to Oracle's Advanced Queuing.

See Also:

- *Oracle9i Application Developer's Guide - Advanced Queuing*
- Chapter 106, "Advanced Queuing Types" for information about the TYPEs to use with DBMS_AQ

This chapter discusses the following topics:

Java Classes

Java interfaces are available for DBMS_AQ and DBMS_AQADM. The Java interfaces are provided in the $ORACLE_HOME/rdbms/jlib/aqapi.jar. Users are required to have EXECUTE privileges on the DBMS_AQIN package to use these interfaces.

Enumerated Constants

When using enumerated constants such as BROWSE, LOCKED, or REMOVE, the PL/SQL constants must be specified with the scope of the packages defining it. All types associated with the operational interfaces have to be prepended with DBMS_AQ. For example: DBMS_AQ.BROWSE.

Table 5–1 Enumerated Constants

Parameter	Options
visibility	IMMEDIATE, ON_COMMIT
dequeue mode	BROWSE, LOCKED, REMOVE, REMOVE_NODATA
navigation	FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION
state	WAITING, READY, PROCESSED, EXPIRED
sequence_deviation	BEFORE, TOP
wait	FOREVER, NO_WAIT
delay	NO_DELAY
expiration	NEVER
namespace	NAMESPACE_AQ, NAMESPACE_ANONYMOUS

Data Structures for DBMS_AQ

Table 5–2 Data Structures for DBMS_AQ

Data Structures
Object Name on page 5-3
Type Name on page 5-3
AQ PL/SQL Callback on page 5-4

Object Name

The `object_name` data structure names database objects. It applies to queues, queue tables, agent names, and object types.

Syntax

```
object_name := VARCHAR2;
object_name := [<schema_name>.]<name>;
```

Usage Notes

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, the current schema is assumed. The name must follow object name guidelines in the *Oracle9i SQL Reference* with regard to reserved characters. Schema names, agent names, and object type names can be up to 30 bytes long. Queue names and queue table names can be up to 24 bytes long.

Type Name

The `type_name` data structure defines queue types.

Syntax

```
type_name := VARCHAR2;
type_name := <object_type> | "RAW";
```

Attributes

Table 5–3 Type Name Attributes

Attribute	Description
`<object_types>`	Maximum number of attributes in the object type is limited to 900.

Table 5–3 (Cont.) Type Name Attributes

Attribute	Description
"RAW"	To store payload of type RAW, AQ creates a queue table with a LOB column as the payload repository. The theoretical maximum size of the message payload is the maximum amount of data that can be stored in a LOB column. However, the maximum size of the payload is determined by which programmatic environment you use to access AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G. Because the PL/SQL enqueue and dequeue interfaces accept RAW buffers as the payload parameters you will be limited to 32K bytes. In OCI, the maximum size of your RAW data will be limited to the maximum amount of contiguous memory (as an OCIRaw is simply an array of bytes) that the OCI Object Cache can allocate. Typically, this will be at least 32K bytes and much larger in many cases. Because LOB columns are used for storing RAW payload, the AQ administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the storage_clause parameter during queue table creation time.

AQ PL/SQL Callback

The plsqlcallback data structure specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification.

Syntax

If a notification message is expected for a RAW payload enqueue, then the PL/SQL callback must have the following signature:

```
procedure plsqlcallback(
  context  IN  RAW,
  reginfo  IN  SYS.AQ$_REG_INFO,
  descr    IN  SYS.AQ$_DESCRIPTOR,
  payload  IN  RAW,
  payloadl IN  NUMBER);
```

Attributes

Table 5–4 AQ PL/SQL Callback Attributes

Attribute	Description
`context`	Specifies the context for the callback function that was passed by `dbms_aq.register`. See "AQ$_REG_INFO Type" on page 106-5.
`reginfo`	See "AQ$_REG_INFO Type" on page 106-5.
`descr`	See "AQ$_DESCRIPTOR Type" on page 106-3.
`payload`	If a notification message is expected for a raw payload enqueue then this contains the raw payload that was enqueued into a non persistent queue. In case of a persistent queue with raw payload this parameter will be null.
`payloadl`	Specifies the length of `payload`. If `payload` is null, `payload1` = 0.

If the notification message is expected for an ADT payload enqueue, the PL/SQL callback must have the following signature:

```
procedure plsqlcallback(
  context  IN RAW,
  reginfo  IN SYS.AQ$_REG_INFO,
  descr    IN SYS.AQ$_DESCRIPTOR,
  payload  IN VARCHAR2,
  payloadl IN NUMBER);
```

Summary of DBMS_AQ Subprograms

Table 5–5 DBMS_AQ Package Subprograms

Subprograms	Description
ENQUEUE Procedure on page 5-6	Adds a message to the specified queue.
DEQUEUE Procedure on page 5-8	Dequeues a message from the specified queue.
LISTEN Procedure on page 5-11	Listen to one or more queues on behalf of a list of agents.

Table 5–5 (Cont.) DBMS_AQ Package Subprograms

Subprograms	Description
REGISTER Procedure on page 5-12	Registers for message notifications
UNREGISTER Procedure on page 5-13	Unregisters a subscription which turns off notification
POST Procedure on page 5-13	Posts to a anonymous subscription which allows all clients who are registered for the subscription to get notifications.
BIND_AGENT Procedure on page 5-14	Creates an entry for an AQ agent in the LDAP directory
UNBIND_AGENT Procedure on page 5-15	Removes an entry for an AQ agent from the LDAP directory

Note: The DBMS_AQ package does not have a purity level defined; therefore, you cannot call any procedure in this package from other procedures that have RNDS, WNDS, RNPS or WNPS constraints defined.

ENQUEUE Procedure

This procedure adds a message to the specified queue.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name           IN      VARCHAR2,
   enqueue_options      IN      enqueue_options_t,
   message_properties   IN      message_properties_t,
   payload              IN      "<type_name>",
   msgid                OUT     RAW);
```

Parameters

Table 5–6 ENQUEUE Procedure Parameters

Parameter	Description
queue_name	Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.

Table 5–6 (Cont.) ENQUEUE Procedure Parameters

Parameter	Description
enqueue_options	See "ENQUEUE_OPTIONS_T Type" on page 106-10.
message_properties	See "MESSAGE_PROPERTIES_T Type" on page 106-11. See "Using Secure Queues" on page 5-7.
payload	Not interpreted by Oracle AQ. The payload must be specified according to the specification in the associated queue table. NULL is an acceptable parameter. For the definition of <type_name> please refer to "Type Name" on page 5-3.
msgid	System generated identification of the message. This is a globally unique identifier that can be used to identify the message at dequeue time.

Usage Notes

The sequence_deviation parameter in enqueue_options can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the enqueue_options parameter relative_msgid. The relationship is identified by the sequence_deviation parameter.

Specifying sequence_deviation for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

If a message is enqueued to a multiconsumer queue with no recipient, and if the queue has no subscribers (or rule-based subscribers that match this message), then the Oracle error ORA_24033 is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

Using Secure Queues

For secure queues, you must specify the sender_id in the messages_properties parameter. See "MESSAGE_PROPERTIES_T Type" on page 106-11 for more information about sender_id.

When you use secure queues, the following are required:

- You must have created a valid AQ Agent using DBMS_AQADM.CREATE_AQ_AGENT. See "CREATE_AQ_AGENT Procedure" on page 6-28.
- You must map sender_id to a database user with enqueue privileges on the secure queue. Use DBMS_AQADM.ENABLE_DB_ACCESS to do this. See "ENABLE_DB_ACCESS Procedure" on page 6-31.

See Also: *Oracle9i Streams* for information about secure queues

DEQUEUE Procedure

This procedure dequeues a message from the specified queue.

Syntax

```
DBMS_AQ.DEQUEUE (
   queue_name           IN      VARCHAR2,
   dequeue_options      IN      dequeue_options_t,
   message_properties   OUT     message_properties_t,
   payload              OUT     "<type_name>",
   msgid                OUT     RAW);
```

Parameters

Table 5–7 DEQUEUE Procedure Parameters

Parameter	Description
queue_name	Specifies the name of the queue.
dequeue_options	See "DEQUEUE_OPTIONS_T Type" on page 106-8. See "Using Secure Queues" on page 5-10.
message_properties	See "MESSAGE_PROPERTIES_T Type" on page 106-11.
payload	Not interpreted by Oracle AQ. The payload must be specified according to the specification in the associated queue table. For the definition of <type_name> please refer to "Type Name" on page 5-3.
msgid	System generated identification of the message.

Usage Notes

The search criteria for messages to be dequeued is determined by the consumer_name, msgid, correlation and deq_condition parameters in dequeue_options.

- Msgid uniquely identifies the message to be dequeued.
- Correlation identifiers are application-defined identifiers that are not interpreted by AQ.
- Dequeue condition is an expression based on the message properties, the message data properties and PL/SQL functions. A deq_condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Message properties include priority, corrid and other columns in the queue table.

  To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload.

  Example: tab.user_data.orderstatus='EXPRESS''

Only messages in the READY state are dequeued unless msgid is specified.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the msgid and correlation ID in dequeue_options.

The database-consistent read mechanism is applicable for queue operations. For example, a BROWSE call may not see a message that is enqueued after the beginning of the browsing transaction.

The default NAVIGATION parameter during dequeue is NEXT_MESSAGE. This means that subsequent dequeues will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command will be processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the FIRST_MESSAGE navigation option when the first message in the queue needs to be processed by every dequeue command. This usually becomes necessary when a

higher priority message arrives in the queue while messages already-enqueued are being processed.

> **Note:** It may be more efficient to use the FIRST_MESSAGE navigation option when messages are concurrently enqueued. If the FIRST_MESSAGE option is not specified, AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the FIRST_MESSAGE option is specified, then AQ uses a new snapshot for every dequeue command.

Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, then this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in LOCKED or REMOVE mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the NEXT_TRANSACTION to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time-out after the specified WAIT period.

Using Secure Queues

For secure queues, you must specify consumer_name in the dequeue_options parameter. See "DEQUEUE_OPTIONS_T Type" on page 106-8 for more information about consumer_name.

When you use secure queues, the following are required:

- You must have created a valid AQ Agent using DBMS_AQADM.CREATE_AQ_AGENT. See "CREATE_AQ_AGENT Procedure" on page 6-28.
- You must map the AQ Agent to a database user with dequeue privileges on the secure queue. Use DBMS_AQADM.ENABLE_DB_ACCESS to do this. See "ENABLE_DB_ACCESS Procedure" on page 6-31.

> **See Also:** *Oracle9i Streams* for information about secure queues

LISTEN Procedure

This procedure listens on one or more queues on behalf of a list of agents. The address field of the agent indicates the queue the agent wants to monitor. Only local queues are supported as addresses. Protocol is reserved for future use.

If agent-address is a multiconsumer queue, then agent-name is mandatory. For single-consumer queues, agent-name must not be specified.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If no messages are found when the wait time expires, an error is raised.

Syntax

```
DBMS_AQ.LISTEN (
   agent_list IN    aq$_agent_list_t,
   wait       IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
   agent      OUT   sys.aq$_agent);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

Parameters

Table 5–8 LISTEN Procedure Parameters

Parameter	Description
agent_list	List of agents to listen for.
wait	Time-out for the listen call (in seconds). By default, the call will block forever.
agent	Agent with a message available for consumption.

Usage Notes

This procedure takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first

agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the listen call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent must still dequeue the relevant message.

Note that you cannot call listen on nonpersistent queues.

REGISTER Procedure

This procedure registers an email address, user-defined PL/SQL procedure, or HTTP URL for message notification.

Syntax

```
DBMS_AQ.REGISTER (
   reg_list IN SYS.AQ$_REG_INFO_LIST,
   count    IN NUMBER);
```

Parameters

Table 5–9 REGISTER Procedure Parameters

Parameter	Description
reg_list	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.
count	Specifies the number of entries in the reg_list.

Usage Notes

This procedure is used to register for notifications. You can specify an email address to which message notifications are sent, register a procedure to be invoked on a notification, or register an HTTP URL to which the notification is posted. Interest in several subscriptions can be registered at one time.

If you register for email notifications, you should set the host name and port name for the SMTP server that will be used by the database to send email notifications. If required, you should set the send-from email address, which is set by the database as the `sent from` field. See Chapter 7, "DBMS_AQELM" for more information on email notifications. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you may want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications. See Chapter 7, "DBMS_AQELM" for more information on HTTP notifications.

UNREGISTER Procedure

This procedure unregisters a subscription which turns off notifications.

Syntax

```
DBMS_AQ.UNREGISTER (
   reg_list IN SYS.AQ$_REG_INFO_LIST,
   count    IN NUMBER);
```

Parameters

Table 5–10 UNREGISTER Procedure Parameters

Parameter	Description
reg_list	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.
count	Specifies the number of entries in the reg_list.

Usage Notes

This procedure is used to unregister a subscription which turns off notifications. Several subscriptions can be unregistered from at one time.

POST Procedure

This procedure posts to a list of anonymous subscriptions that allows all clients who are registered for the subscriptions to get notifications.

Syntax

```
DBMS_AQ.POST (
  post_list  IN  SYS.AQ$_POST_INFO_LIST,
  count      IN NUMBER);
```

Parameters

Table 5–11 POST Procedure Parameters

Parameter	Description
post_list	Specifies the list of anonymous subscriptions to which you want to post. It is a list of AQ$_POST_INFO Type.
count	Specifies the number of entries in the post_list.

Usage Notes

This procedure is used to post to anonymous subscriptions which allows all clients who are registered for the subscriptions to get notifications. Several subscriptions can be posted to at one time.

BIND_AGENT Procedure

This procedure creates an entry for an AQ agent in the LDAP server.

Syntax

```
DBMS_AQ.BIND_AGENT(
   agent        IN SYS.AQ$_AGENT,
   certificate  IN VARCHAR2 default NULL);
```

Parameters

Table 5–12 BIND_AGENT Procedure Parameters

Parameter	Description
agent	Agent that is to be registered in LDAP server
certificate	Location (LDAP distinguished name) of the "organizationalperson" entry in LDAP whose digital certificate (attribute usercertificate) is to be used for this agent Example: "cn=OE, cn=ACME, cn=com" is a DN for a OrganizationalPerson OE whose certificate will be used with the specified agent.

Usage Notes

In the LDAP server, digital certificates are stored as an attribute (`usercertificate`) of the `OrganizationalPerson` entity. The distinguished name for this `OrganizationalPerson` must be specified when binding the agent.

UNBIND_AGENT Procedure

This procedure removes the entry for an AQ agent from the LDAP server.

Syntax

```
DBMS_AQ.UNBIND_AGENT(
   agent     IN SYS.AQ$_AGENT);
```

Parameters

Table 5–13 BIND_AGENT Procedure Parameters

Parameter	Description
`agent`	Agent that is to be removed from the LDAP server

6

DBMS_AQADM

The DBMS_AQADM package provides procedures to manage Advanced Queuing configuration and administration information.

See Also:

- *Oracle9i Application Developer's Guide - Advanced Queuing*
- Chapter 106, "Advanced Queuing Types" for information about the TYPEs to use with DBMS_AQADM

This chapter discusses the following topics:

Enumerated Constants

When using enumerated constants, such as INFINITE, TRANSACTIONAL, or NORMAL_QUEUE, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with DBMS_AQADM. For example: DBMS_AQADM.NORMAL_QUEUE.

Table 6–1 Enumerated Types in the Administrative Interface

Parameter	Options
retention	0,1,2...INFINITE
message_grouping	TRANSACTIONAL, NONE
queue_type	NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE

See Also: For more information on the Java classes and data structures used in both DBMS_AQ and DBMS_AQADM, see Chapter 5, "DBMS_AQ"

Summary of DBMS_AQADM Subprograms

Table 6–2 DBMS_AQADM Package Subprograms

Subprogram	Description
CREATE_QUEUE_TABLE Procedure on page 6-4	Creates a queue table for messages of a predefined type.
ALTER_QUEUE_TABLE Procedure on page 6-8	Alters an existing queue table.
DROP_QUEUE_TABLE Procedure on page 6-9	Drops an existing queue table.
CREATE_QUEUE Procedure on page 6-9	Creates a queue in the specified queue table.
CREATE_NP_QUEUE Procedure on page 6-11	Creates a nonpersistent RAW queue.
ALTER_QUEUE Procedure on page 6-12	Alters existing properties of a queue.
DROP_QUEUE Procedure on page 6-14	Drops an existing queue.

Table 6–2 (Cont.) DBMS_AQADM Package Subprograms

Subprogram	Description
START_QUEUE Procedure on page 6-14	Enables the specified queue for enqueuing or dequeueing.
STOP_QUEUE Procedure on page 6-15	Disables enqueuing or dequeuing on the specified queue.
GRANT_SYSTEM_PRIVILEGE Procedure on page 6-16	Grants AQ system privileges to users and roles.
REVOKE_SYSTEM_PRIVILEGE Procedure on page 6-17	Revokes AQ system privileges from users and roles.
GRANT_QUEUE_PRIVILEGE Procedure on page 6-18	Grants privileges on a queue to users and roles.
REVOKE_QUEUE_PRIVILEGE Procedure on page 6-19	Revokes privileges on a queue from users and roles.
ADD_SUBSCRIBER Procedure on page 6-19	Adds a default subscriber to a queue.
ALTER_SUBSCRIBER Procedure on page 6-21	Alters existing properties of a subscriber to a specified queue.
REMOVE_SUBSCRIBER Procedure on page 6-21	Removes a default subscriber from a queue.
SCHEDULE_PROPAGATION Procedure on page 6-22	Schedules propagation of messages from a queue to a destination identified by a specific dblink.
UNSCHEDULE_ PROPAGATION Procedure on page 6-24	Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific dblink.
VERIFY_QUEUE_TYPES Procedure on page 6-24	Verifies that the source and destination queues have identical types.
ALTER_PROPAGATION_ SCHEDULE Procedure on page 6-25	Alters parameters for a propagation schedule.
ENABLE_PROPAGATION_ SCHEDULE Procedure on page 6-27	Enables a previously disabled propagation schedule.
DISABLE_PROPAGATION_ SCHEDULE Procedure on page 6-27	Disables a propagation schedule.

Table 6–2 (Cont.) DBMS_AQADM Package Subprograms

Subprogram	Description
MIGRATE_QUEUE_TABLE Procedure on page 6-28	Upgrades an 8.0-compatible queue table to an 8.1-compatible queue table, or downgrades an 8.1-compatible queue table to an 8.0-compatible queue table.
CREATE_AQ_AGENT Procedure on page 6-28	Registers an agent for AQ Internet access
ALTER_AQ_AGENT Procedure on page 6-29	Alters an agent registered for AQ Internet access
DROP_AQ_AGENT Procedure on page 6-30	Drops an agent registered for AQ Internet access
ENABLE_DB_ACCESS Procedure on page 6-31	Grants an AQ Internet agent the privileges of a specific database user
DISABLE_DB_ACCESS Procedure on page 6-32	Revokes the privileges of a database user from an AQ Internet agent
ADD_ALIAS_TO_LDAP Procedure on page 6-32	Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP.
DEL_ALIAS_FROM_LDAP Procedure on page 6-33	Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP.

CREATE_QUEUE_TABLE Procedure

This procedure creates a queue table for messages of a predefined type. The sort keys for dequeue ordering, if any, must be defined at table creation time. The following objects are created at this time:

- A default exception queue associated with the queue table, called `aq$_<queue_table_name>_e`.
- A read-only view, which is used by AQ applications for querying queue data, called `aq$<queue_table_name>`.
- An index or an index organized table (IOT) in the case of multiple consumer queues for the queue monitor operations, called `aq$_<queue_table_name>_t`.
- An index or an index organized table in the case of multiple consumer queues for dequeue operations, called `aq$_<queue_table_name>_i`.

For Oracle8*i*-compatible queue tables, the following index-organized tables are created:

- A table called aq$_<queue_table_name>_s. This table stores information about the subscribers.
- A table called aq$_<queue_table_name>_r. This table stores information about rules on subscriptions.
- An index-organized table called aq$_<queue_table_name>_h. This table stores the dequeue history data.

Syntax

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table          IN      VARCHAR2,
   queue_payload_type   IN      VARCHAR2,
   storage_clause       IN      VARCHAR2       DEFAULT NULL,
   sort_list            IN      VARCHAR2       DEFAULT NULL,
   multiple_consumers   IN      BOOLEAN        DEFAULT FALSE,
   message_grouping     IN      BINARY_INTEGER DEFAULT NONE,
   comment              IN      VARCHAR2       DEFAULT NULL,
   auto_commit          IN      BOOLEAN        DEFAULT TRUE,
   primary_instance     IN      BINARY_INTEGER DEFAULT 0,
   secondary_instance   IN      BINARY_INTEGER DEFAULT 0,
   compatible           IN      VARCHAR2       DEFAULT NULL);
```

Parameters

Table 6–3 CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be created.
queue_payload_type	Type of the user data stored. See "Type Name" Chapter 5, "DBMS_AQ" for valid values for this parameter.

Table 6–3 (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
storage_clause	Storage parameter. The storage parameter is included in the CREATE TABLE statement when the queue table is created. The storage parameter can be made up of any combinations of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLEPSACE, LOB, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See *Oracle9i SQL Reference* for the usage of these parameters.
sort_list	The columns to be used as the sort key in ascending order. Sort_list has the following format: '<sort_column_1>,<sort_column_2>' The allowed column names are priority and enq_time. If both columns are specified, then <sort_column_1> defines the most significant order. After a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same defaults. The order of a queue table cannot be altered after the queue table has been created. If no sort list is specified, then all the queues in this queue table are sorted by the enqueue time in ascending order. This order is equivalent to FIFO order. Even with the default ordering defined, a dequeuer is allowed to choose a message to dequeue by specifying its msgid or correlation. msgid, correlation, and sequence_deviation take precedence over the default dequeueing order, if they are specified.
multiple_consumers	FALSE: Queues created in the table can only have one consumer for each message. This is the default. TRUE: Queues created in the table can have multiple consumers for each message.

Table 6–3 (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
message_grouping	Message grouping behavior for queues created in the table. NONE: Each message is treated individually. TRANSACTIONAL: Messages enqueued as part of one transaction are considered part of the same group and can be dequeued as a group of related messages.
comment	User-specified description of the queue table. This user comment is added to the queue catalog.
auto_commit	TRUE: causes the current transaction, if any, to commit before the CREATE_QUEUE_TABLE operation is carried out. The CREATE_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Note: This parameter has been deprecated.
primary_instance	The primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value for primary instance is 0, which means queue monitor scheduling and propagation will be done in any available instance.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is 0, which means that the queue table will fail over to any available instance.
compatible	The lowest database version with which the queue is compatible. Currently the possible values are either '8.0' or '8.1'. ■ If the database is in 8.1 or higher compatible mode, the default value is '8.1' ■ If the database is in 8.0 compatible mode, the default value is '8.0'

Usage Notes

CLOB, BLOB, and BFILE are valid attributes for AQ object type payloads. However, only CLOB and BLOB can be propagated using AQ propagation in Oracle8*i* release 8.1.5 or later. See the *Oracle9i Application Developer's Guide - Advanced Queuing* for more information.

The default value of the compatible parameter depends on the database compatibility mode in the `init.ora`.

- If the database is in 8.1 or higher compatible mode, the default value is 8.1
- If the database is in 8.0 compatible mode, the default value is 8.0

You can specify and modify the `primary_instance` and `secondary_instance` only in 8.1-compatible mode.

You cannot specify a secondary instance unless there is a primary instance.

ALTER_QUEUE_TABLE Procedure

This procedure alters the existing properties of a queue table.

Syntax

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
   queue_table          IN   VARCHAR2,
   comment              IN   VARCHAR2       DEFAULT NULL,
   primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
   secondary_instance   IN   BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 6–4 ALTER_QUEUE_TABLE Procedure Parameters

Parameter	Description
`queue_table`	Name of a queue table to be created.
`comment`	Modifies the user-specified description of the queue table. This user comment is added to the queue catalog. The default value is `NULL` which means that the value will not be changed.
`primary_instance`	This is the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table will be done in this instance. The default value is `NULL`, which means that the current value will not be changed.
`secondary_instance`	The queue table fails over to the secondary instance if the primary instance is not available. The default value is `NULL`, which means that the current value will not be changed.

DROP_QUEUE_TABLE Procedure

This procedure drops an existing queue table. All the queues in a queue table must be stopped and dropped before the queue table can be dropped. You must do this explicitly unless the `force` option is used, in which case this is done automatically.

Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table       IN    VARCHAR2,
   force             IN    BOOLEAN DEFAULT FALSE,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 6–5 DROP_QUEUE_TABLE Procedure Parameters

Parameter	Description
`queue_table`	Name of a queue table to be dropped.
`force`	FALSE: The operation does not succeed if there are any queues in the table. This is the default. TRUE: All queues in the table are stopped and dropped automatically.
`auto_commit`	TRUE: Causes the current transaction, if any, to commit before the DROP_QUEUE_TABLE operation is carried out. The DROP_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

CREATE_QUEUE Procedure

This procedure creates a queue in the specified queue table.

Syntax

```
DBMS_AQADM.CREATE_QUEUE (
   queue_name          IN       VARCHAR2,
   queue_table         IN       VARCHAR2,
   queue_type          IN       BINARY_INTEGER DEFAULT NORMAL_QUEUE,
   max_retries         IN       NUMBER         DEFAULT NULL,
   retry_delay         IN       NUMBER         DEFAULT 0,
```

```
retention_time      IN      NUMBER          DEFAULT 0,
dependency_tracking IN      BOOLEAN         DEFAULT FALSE,
comment             IN      VARCHAR2        DEFAULT NULL,
auto_commit         IN      BOOLEAN         DEFAULT TRUE);
```

Parameters

Table 6–6 CREATE_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in *Oracle9i SQL Reference* with regard to reserved characters.
queue_table	Name of the queue table that will contain the queue.
queue_type	Specifies whether the queue being created is an exception queue or a normal queue. NORMAL_QUEUE: The queue is a normal queue. This is the default. EXCEPTION_QUEUE: It is an exception queue. Only the dequeue operation is allowed on the exception queue.
max_retries	Limits the number of times a dequeue with the REMOVE mode can be attempted on a message. The maximum value of max_retries is 2**31 -1. The count is incremented when the application issues a rollback after executing the dequeue. The message is moved to the exception queue when it is reaches its max_retries. Note that max_retries is supported for all single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time, in seconds, before this message is scheduled for processing again after an application rollback. The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if max_retries is set to 0. Note that rety_delay is supported for single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.

Table 6–6 (Cont.) CREATE_QUEUE Procedure Parameters

Parameter	Description
retention_time	Number of seconds for which a message is retained in the queue table after being dequeued from the queue. INFINITE: Message is retained forever. NUMBER: Number of seconds for which to retain the messages. The default is 0, no retention.
dependency_tracking	Reserved for future use. FALSE: This is the default. TRUE: Not permitted in this release.
comment	User-specified description of the queue. This user comment is added to the queue catalog.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the CREATE_QUEUE operation is carried out. The CREATE_QUEUE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Usage Notes

All queue names must be unique within a schema. After a queue is created with CREATE_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

CREATE_NP_QUEUE Procedure

Create a nonpersistent RAW queue.

Syntax

```
DBMS_AQADM.CREATE_NP_QUEUE (
   queue_name              IN       VARCHAR2,
   multiple_consumers      IN       BOOLEAN  DEFAULT FALSE,
   comment                 IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–7 CREATE_NP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the nonpersistent queue that is to be created. The name must be unique within a schema and must follow object name guidelines in *Oracle9i SQL Reference.*
multiple_consumers	FALSE: Queues created in the table can only have one consumer for each message. This is the default. TRUE: Queues created in the table can have multiple consumers for each message. Note that this parameter is distinguished at the queue level, because a nonpersistent queue does not inherit this characteristic from any user-created queue table.
comment	User-specified description of the queue. This user comment is added to the queue catalog.

Usage Notes

The queue may be either single-consumer or multiconsumer queue. All queue names must be unique within a schema. The queues are created in a 8.1-compatible system-created queue table (AQ$_MEM_SC or AQ$_MEM_MC) in the same schema as that specified by the queue name.

If the queue name does not specify a schema name, the queue is created in the login user's schema. After a queue is created with CREATE_NP_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the OCI notification mechanism. You cannot invoke the `listen` call on a nonpersistent queue.

ALTER_QUEUE Procedure

This procedure alters existing properties of a queue. The parameters `max_retries`, `retention_time`, and `retry_delay` are not supported for nonpersistent queues.

Syntax

```
DBMS_AQADM.ALTER_QUEUE (
   queue_name        IN    VARCHAR2,
```

```
   max_retries       IN    NUMBER   DEFAULT NULL,
   retry_delay       IN    NUMBER   DEFAULT NULL,
   retention_time    IN    NUMBER   DEFAULT NULL,
   auto_commit       IN    BOOLEAN  DEFAULT TRUE,
   comment           IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–8 ALTER_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be altered.
max_retries	Limits the number of times a dequeue with REMOVE mode can be attempted on a message. The maximum value of max_retries is 2**31 -1. The count is incremented when the application issues a rollback after executing the dequeue. If the time at which one of the retries has passed the expiration time, then no further retries are attempted. Default is NULL, which means that the value will not be altered. Note that max_retries is supported for all single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time in seconds before this message is scheduled for processing again after an application rollback. The default is NULL, which means that the value will not be altered. Note that retry_delay is supported for single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retention_time	Retention time in seconds for which a message is retained in the queue table after being dequeued. The default is NULL, which means that the value will not be altered.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_QUEUE operation become persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Table 6–8 (Cont.) ALTER_QUEUE Procedure Parameters

Parameter	Description
comment	User-specified description of the queue. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.

DROP_QUEUE Procedure

This procedure drops an existing queue. DROP_QUEUE is not allowed unless STOP_QUEUE has been called to disable the queue for both enqueuing and dequeuing. All the queue data is deleted as part of the drop operation.

Syntax

```
DBMS_AQADM.DROP_QUEUE (
   queue_name        IN    VARCHAR2,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 6–9 DROP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be dropped.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the DROP_QUEUE operation is carried out. The DROP_QUEUE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

START_QUEUE Procedure

This procedure enables the specified queue for enqueuing or dequeueing.

After creating a queue, the administrator must use START_QUEUE to enable the queue. The default is to enable it for both ENQUEUE and DEQUEUE. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Syntax

```
DBMS_AQADM.START_QUEUE (
   queue_name      IN     VARCHAR2,
   enqueue         IN     BOOLEAN DEFAULT TRUE,
   dequeue         IN     BOOLEAN DEFAULT TRUE);
```

Parameters

Table 6–10 START_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be enabled.
enqueue	Specifies whether ENQUEUE should be enabled on this queue. TRUE: Enable ENQUEUE. This is the default. FALSE: Do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be enabled on this queue. TRUE: Enable DEQUEUE. This is the default. FALSE: Do not alter the current setting.

STOP_QUEUE Procedure

This procedure disables enqueuing or dequeuing on the specified queue.

By default, this call disables both ENQUEUEs or DEQUEUEs. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Syntax

```
DBMS_AQADM.STOP_QUEUE (
   queue_name      IN   VARCHAR2,
   enqueue         IN   BOOLEAN DEFAULT TRUE,
   dequeue         IN   BOOLEAN DEFAULT TRUE,
   wait            IN   BOOLEAN DEFAULT TRUE);
```

Parameters

Table 6–11 STOP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be disabled.
enqueue	Specifies whether ENQUEUE should be disabled on this queue. TRUE: Disable ENQUEUE. This is the default. FALSE: Do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be disabled on this queue. TRUE: Disable DEQUEUE. This is the default. FALSE: Do not alter the current setting.
wait	Specifies whether to wait for the completion of outstanding transactions. TRUE: Wait if there are any outstanding transactions. In this state no new transactions are allowed to enqueue to or dequeue from this queue. FALSE: Return immediately either with a success or an error.

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grants AQ system privileges to users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. Initially, only SYS and SYSTEM can use this procedure successfully.

Syntax

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
   privilege         IN    VARCHAR2,
   grantee           IN    VARCHAR2,
   admin_option      IN    BOOLEAN := FALSE);
```

Parameters

Table 6–12 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ system privilege to grant. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The operations allowed for each system privilege are specified as follows: ENQUEUE_ANY: users granted with this privilege are allowed to enqueue messages to any queues in the database. DEQUEUE_ANY: users granted with this privilege are allowed to dequeue messages from any queues in the database. MANAGE_ANY: users granted with this privilege are allowed to run DBMS_AQADM calls on any schemas in the database.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
admin_option	Specifies if the system privilege is granted with the ADMIN option or not. If the privilege is granted with the ADMIN option, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles. The default is FALSE.

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes AQ system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.

Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (
   privilege         IN   VARCHAR2,
   grantee           IN   VARCHAR2);
```

Parameters

Table 6–13 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ system privilege to revoke. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.

GRANT_QUEUE_PRIVILEGE Procedure

This procedure grants privileges on a queue to users and roles. The privileges are ENQUEUE or DEQUEUE. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Syntax

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege        IN    VARCHAR2,
   queue_name       IN    VARCHAR2,
   grantee          IN    VARCHAR2,
   grant_option     IN    BOOLEAN := FALSE);
```

Parameters

Table 6–14 GRANT_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
grant_option	Specifies if the access privilege is granted with the GRANT option or not. If the privilege is granted with the GRANT option, then the grantee is allowed to use this procedure to grant the access privilege to other users or roles, regardless of the ownership of the queue table. The default is FALSE.

REVOKE_QUEUE_PRIVILEGE Procedure

This procedure revokes privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE. To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the GRANT option are revoked if the grantor's privileges are revoked.

Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege         IN      VARCHAR2,
   queue_name        IN      VARCHAR2,
   grantee           IN      VARCHAR2);
```

Parameters

Table 6–15 REVOKE_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ queue privilege to revoke. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role. If the privilege has been propagated by the grantee through the GRANT option, then the propagated privilege is also revoked.

ADD_SUBSCRIBER Procedure

This procedure adds a default subscriber to a queue.

Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2 DEFAULT NULL,
   transformation IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–16 ADD_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being defined.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. A rule is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Currently supported message properties are priority and corrid. To specify rules on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The rule parameter cannot exceed 4000 characters.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue

Usage Notes

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation only succeeds on queues that allow multiple consumers. This operation takes effect immediately, and the containing transaction is committed. Enqueue requests that are executed after the completion of this call will reflect the new behavior.

Any string within the rule must be quoted:

```
rule   => 'PRIORITY <= 3 AND CORRID =  ''FROM JAPAN'''
```

Note that these are all single quotation marks.

ALTER_SUBSCRIBER Procedure

This procedure alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

Syntax

```
DBMS_AQADM.ALTER_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2
   transformation IN    VARCHAR2);
```

Parameters

Table 6–17 ALTER_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being altered. See "AQ$_AGENT Type" on page 106-2.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. Note: The rule parameter cannot exceed 4000 characters. To eliminate the rule, set the rule parameter to NULL.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue

Usage Notes

This procedure alters both the rule and the transformation for the subscriber. If you want to retain the existing value for either of them, you must specify its old value. The current values for rule and transformation for a subscriber can be obtained from the <schema>.AQ$<queue_table>_R and <schema>.AQ$<queue_table>_S views.

REMOVE_SUBSCRIBER Procedure

This procedure removes a default subscriber from a queue. This operation takes effect immediately, and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation.

Syntax

```
DBMS_AQADM.REMOVE_SUBSCRIBER (
   queue_name          IN          VARCHAR2,
   subscriber          IN          sys.aq$_agent);
```

Parameters

Table 6–18 REMOVE_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent who is being removed. See "AQ$_AGENT Type" on page 106-2.

SCHEDULE_PROPAGATION Procedure

This procedure schedules propagation of messages from a queue to a destination identified by a specific dblink.

Messages may also be propagated to other queues in the same database by specifying a NULL destination. If a message has multiple recipients at the same destination in either the same or different queues, then the message is propagated to all of them at the same time.

Syntax

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
   queue_name       IN    VARCHAR2,
   destination      IN    VARCHAR2 DEFAULT NULL,
   start_time       IN    DATE     DEFAULT SYSDATE,
   duration         IN    NUMBER   DEFAULT NULL,
   next_time        IN    VARCHAR2 DEFAULT NULL,
   latency          IN    NUMBER   DEFAULT 60);
```

Parameters

Table 6–19 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
start_time	Initial start time for the propagation window for messages from the source queue to the destination.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as 'SYSDATE + 1 - duration/86400'.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. For example: If the latency is 60 seconds, then during the propagation window; if there are no messages to be propagated, then messages from that queue for the destination are not propagated for at least 60 more seconds. It is at least 60 seconds before the queue is checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue is not checked for 10 minutes, and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination. As soon as a message is enqueued, it is propagated.

UNSCHEDULE_PROPAGATION Procedure

This procedure unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific dblink.

Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
   queue_name      IN   VARCHAR2,
   destination     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–20 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

VERIFY_QUEUE_TYPES Procedure

This procedure verifies that the source and destination queues have identical types. The result of the verification is stored in the table sys.aq$_message_types, overwriting all previous output of this command.

Syntax

```
DBMS_AQADM.VERIFY_QUEUE_TYPES (
   src_queue_name    IN    VARCHAR2,
   dest_queue_name   IN    VARCHAR2,
   destination       IN    VARCHAR2 DEFAULT NULL,
   rc                OUT   BINARY_INTEGER);
```

Parameters

Table 6–21 VERIFY_QUEUE_TYPES Procedure Parameters

Parameter	Description
src_queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
dest_queue_name	Name of the destination queue where messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
rc	Return code for the result of the procedure. If there is no error, and if the source and destination queue types match, then the result is 1. If they do not match, then the result is 0. If an Oracle error is encountered, then it is returned in rc.

ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters parameters for a propagation schedule.

Syntax

```
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE (
   queue_name      IN     VARCHAR2,
   destination     IN     VARCHAR2 DEFAULT NULL,
   duration        IN     NUMBER   DEFAULT NULL,
   next_time       IN     VARCHAR2 DEFAULT NULL,
   latency         IN     NUMBER   DEFAULT 60);
```

Parameters

Table 6–22 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as 'SYSDATE + 1 - duration/86400'.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. The default value is 60. Caution: if latency is not specified for this call, then latency will over-write any existing value with the default value. For example, if the latency is 60 seconds, then during the propagation window, if there are no messages to be propagated, then messages from that queue for the destination will not be propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.

ENABLE_PROPAGATION_SCHEDULE Procedure

This procedure enables a previously disabled propagation schedule.

Syntax

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
   queue_name       IN     VARCHAR2,
   destination      IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–23 ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
   queue_name       IN     VARCHAR2,
   destination      IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–24 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

MIGRATE_QUEUE_TABLE Procedure

This procedure upgrades an 8.0-compatible queue table to an 8.1-compatible queue table, or downgrades an 8.1-compatible queue table to an 8.0-compatible queue table.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (
   queue_table    IN   VARCHAR2,
   compatible     IN   VARCHAR2);
```

Parameters

Table 6–25 MIGRATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Specifies name of the queue table to be migrated.
compatible	Set this to '8.1' to upgrade an 8.0-compatible queue table, or set this to '8.0' to downgrade an 8.1-compatible queue table.

CREATE_AQ_AGENT Procedure

This procedure registers an agent for AQ Internet access using HTTP/SMTP protocols. It is also used to create an AQ agent to access secure queues.

See Also: *Oracle9i Streams* for information about secure queues

Syntax

```
DBMS_AQADM.CREATE_AQ_AGENT (
  agent_name                IN VARCHAR2,
  certificate_location      IN VARCHAR2 DEFAULT NULL,
  enable_http               IN BOOLEAN DEFAULT FALSE,
  enable_smtp               IN BOOLEAN DEFAULT FALSE,
  enable_anyp               IN BOOLEAN DEFAULT FALSE )
```

Parameters

Table 6–26 CREATE_AQ_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the AQ Internet agent
certification_ location	Agent's certificate location in LDAP (default= NULL). If the agent is allowed to access AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required
enable_http	TRUE: the agent can access AQ through HTTP FALSE: the agent cannot access AQ through HTTP
enable_smtp	TRUE: the agent can access AQ through SMTP (e-mail) FALSE: the agent cannot access AQ through SMTP
enable_anyp	TRUE: the agent can access AQ through any protocol (HTTP or SMTP)

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all AQ Internet agents.

ALTER_AQ_AGENT Procedure

This procedure alters an agent registered for AQ Internet access. It is also used to alter an AQ agent that accesses secure queues.

See Also: *Oracle9i Streams* for information about secure queues

Syntax

```
DBMS_AQADM.ALTER_AQ_AGENT (
  agent_name                IN VARCHAR2,
  certificate_location      IN VARCHAR2 DEFAULT NULL,
  enable_http               IN BOOLEAN DEFAULT FALSE,
  enable_smtp               IN BOOLEAN DEFAULT FALSE,
  enable_anyp               IN BOOLEAN DEFAULT FALSE )
```

Parameters

Table 6–27 ALTER_AQ_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the AQ Internet agent
certification_ location	Agent's certificate location in LDAP (default= NULL). If the agent is allowed to access AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required
enable_http	TRUE: the agent can access AQ through HTTP FALSE: the agent cannot access AQ through HTTP
enable_smtp	TRUE: the agent can access AQ through SMTP (e-mail) FALSE: the agent cannot access AQ through SMTP
enable_anyp	TRUE: the agent can access AQ through any protocol (HTTP or SMTP)

DROP_AQ_AGENT Procedure

This procedure drops an agent that was previously registered for AQ Internet access.

Syntax

```
DBMS_AQADM.DROP_AQ_AGENT (
  agent_name                IN VARCHAR2)
```

Parameters

Table 6–28 DROP_AQ_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the AQ Internet agent

ENABLE_DB_ACCESS Procedure

This procedure grants an AQ Internet agent the privileges of a specific database user. The AQ Internet agent should have been previously created using the CREATE_AQ_AGENT procedure.

For secure queues, the sender and receiver agent of the message must be mapped to the database user performing the enqueue or dequeue operation.

See Also: *Oracle9i Streams* for information about secure queues

Syntax

```
DBMS_AQADM.ENABLE_DB_ACCESS (
  agent_name                     IN VARCHAR2,
  db_username                    IN VARCHAR2)
```

Parameters

Table 6–29 ENABLE_DB_ACCESS Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the AQ Internet agent
db_username	Specified the database user whose privileges are to be granted to the AQ Internet agent

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all AQ Internet agents and the names of the database users whose privileges are granted to them.

DISABLE_DB_ACCESS Procedure

This procedure revokes the privileges of a specific database user from an AQ Internet agent. The AQ Internet agent should have been previously granted those privileges using the ENABLE_DB_ACCESS procedure.

Syntax

```
DBMS_AQADM.DISABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

Parameters

Table 6–30 DISABLE_DB_ACCESS Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the AQ Internet agent
db_username	Specified the database user whose privileges are to be revoked from the AQ Internet agent

ADD_ALIAS_TO_LDAP Procedure

This procedure creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP. The alias will be placed directly under the database server's distinguished name in LDAP hierarchy.

Syntax

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP(
   alias          IN VARCHAR2,
   obj_location   IN VARCHAR2);
```

Parameters

Table 6–31 ADD_ALIAS_TO_LDAP Procedure Parameters

Parameter	Description
alias	the name of the alias Example:'west_shipping'
obj_location	The distinguished name of the object (queue, agent or connection factory) to which alias refers

Usage Notes

This method can be used to create aliases for Queues, Agents and JMS ConnectionFactory objects. These object must exist before the alias is created. These aliases can be used for JNDI lookup in JMS and AQ Internet access.

DEL_ALIAS_FROM_LDAP Procedure

This procedure drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP.

Syntax

```
DBMS_AQ.DEL_ALIAS_FROM_LDAP(
   alias IN VARCHAR2);
```

Parameters

Table 6–32 DEL_ALIAS_FROM_LDAP Procedure Parameters

Parameter	Description
alias	The alias to be removed

7

DBMS_AQELM

The DBMS_AQELM package provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP.

> **See Also:** *Oracle9i Application Developer's Guide - Advanced Queuing* for detailed information about DBMS_AQELM.

This chapter discusses the following topics:

Summary of DBMS_AQELM Subprograms

Table 7–1 DBMS_AQELM Subprograms

Subprogram	Description
SET_MAILHOST Procedure on page 7-2	Sets the host name for SMTP server.
GET_MAILHOST Procedure on page 7-3	Gets the host name for SMTP server.
SET_MAILPORT Procedure on page 7-3	Sets the port number for SMTP server.
GET_MAILPORT Procedure on page 7-4	Gets the port number for SMTP server.
SET_SENDFROM Procedure on page 7-4	Sets the sent-from e-mail address.
GET_SENDFROM Procedure on page 7-5	Gets the sent-from e-mail address.
SET_PROXY Procedure on page 7-5	Sets the proxy server name to be used for requests of HTTP protocol, excluding requests for hosts that belong to the domain specified in `no_proxy_domains`.
GET_PROXY Procedure on page 7-6	Gets the proxy server name and `no_proxy_domains` set by `DBMS_AQELM.SET_PROXY` for HTTP notifications.

SET_MAILHOST Procedure

This procedure sets the host name for the SMTP server. As part of the configuration for e-mail notifications, a user with `AQ_ADMINISTRATOR_ROLE` or with `EXECUTE` permissions on the `DBMS_AQELM` package needs to set the host name before registering for e-mail notifications. The database will use this SMTP server host name to send out e-mail notifications.

Syntax

```
DBMS_AQELM.SET_MAILHOST (
   mailhost  IN  VARCHAR2);
```

Parameters

Table 7–2 SET_MAILHOST Procedure Parameters

Parameter	Description
mailhost	The SMTP server host name.

GET_MAILHOST Procedure

This procedure gets the host name set by DBMS_AQELM.SET_MAILHOST for the SMTP server.

Syntax

```
DBMS_AQELM.GET_MAILHOST (
   mailhost  OUT VARCHAR2);
```

Parameters

Table 7–3 GET_MAILHOST Procedure Parameters

Parameter	Description
mailhost	The SMTP server host name.

SET_MAILPORT Procedure

This procedure sets the port number for the SMTP server. As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on DBMS_AQELM package needs to set the port number before registering for e-mail notifications. The database will use this SMTP server port number to send out e-mail notifications. If not set, the SMTP mailport defaults to 25.

Syntax

```
DBMS_AQELM.SET_MAILPORT (
   mailport  IN NUMBER);
```

Parameters

Table 7–4 shows the parameters for the SET_MAILPORT procedure.

Table 7–4 SET_MAILPORT Procedure Parameters

Parameter	Description
mailport	The SMTP server port number.

GET_MAILPORT Procedure

This procedure gets the port number for the SMTP server set by the DBMS_AQELM.SET_MAILPORT procedure or the default value, which is 25.

Syntax

```
DBMS_AQELM.GET_MAILPORT (
   mailport  OUT NUMBER);
```

Parameters

Table 7–5 GET_MAILPORT Procedure Parameters

Parameter	Description
mailport	The SMTP server port number.

SET_SENDFROM Procedure

This procedure sets the sent-from e-mail address. As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package should set the sent-from address before registering for e-mail notifications This e-mail address is used in the sent-from field in all the e-mail notifications sent out by the database to the registered e-mail addresses.

Syntax

```
DBMS_AQELM.SET_SENDFROM (
   sendfrom  IN  VARCHAR2);
```

Parameters

Table 7–6 SET_SENDFROM Procedure Parameters

Parameter	Description
sendfrom	The sent-from e-mail address.

GET_SENDFROM Procedure

This procedure gets the sent-from e-mail address set by DBMS_AQELM.SET_SENDFROM procedure.

Syntax

```
DBMS_AQELM.GET_SENDFROM (
   sendfrom  OUT  VARCHAR2);
```

Parameters

Table 7–7 GET_SENDFROM Procedure Parameters

Parameter	Procedure
sendfrom	The sent-from e-mail address.

SET_PROXY Procedure

This procedure sets the proxy server name to be used for requests of HTTP protocol, excluding requests for hosts that belong to the domain specified in no_proxy_domains. The proxy server name can include an optional TCP/IP port number at which the proxy server listens. If the port is not specified for the proxy server, port 80 is assumed. no_proxy_domains is a list of domains or hosts for which HTTP requests should be sent directly to the destination HTTP server instead of going through a proxy server. Optionally, a port number can be specified for each domain or host. If the port number is specified, the no-proxy restriction is only applied to the request at that port of the particular domain or host. When no_proxy_domains is NULL and the proxy server is set, all requests go through the proxy server. When the proxy server is not set, http_send sends the requests to the target Web servers directly.

As part of the configuration for HTTP notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package can choose to set the proxy server name and a list of no_proxy_domains, if required, before registering for HTTP notifications. The database will use this information to post HTTP notifications.

Syntax

```
DBMS_AQELM.SET_PROXY (
   proxy            IN VARCHAR2,
   no_proxy_domains IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 7–8 SET_PROXY Procedure Parameters

Parameter	Procedure
proxy	The proxy server host and port number. The syntax is "[http://]host[:port][/]". For example, "www-proxy.my-company.com:80".
no_proxy_domains	The list of no-proxy domains or hosts. The syntax is a list of host or domains, with optional port numbers separated by a comma, a semi-colon, or a space. For example, "corp.my-company.com, eng.my-company.com:80"

GET_PROXY Procedure

This procedure gets the proxy server name and no_proxy_domains set by DBMS_AQELM.SET_PROXY for HTTP notifications.

Syntax

```
DBMS_AQELM.GET_PROXY (
   proxy           OUT VARCHAR2,
   no_proxy_domains OUT VARCHAR2);
```

Parameters

Table 7–9 GET_PROXY Procedure Parameters

Parameter	Procedure
proxy	The proxy server host and port number.
no_proxy_domains	The list of no-proxy domains or hosts.

8

DBMS_CAPTURE_ADM

The DBMS_CAPTURE_ADM package provides administrative procedures for starting, stopping, and configuring a capture process. The source of the captured changes is the redo logs, and the repository for the captured changes is a queue (created using the DBMS_AQADM package or the DBMS_STEAMS_ADM.SET_UP_QUEUE procedure).

This chapter contains the following topic:

- Summary of DBMS_CAPTURE_ADM Subprograms

See Also: *Oracle9i Streams* for more information about the capture process

Summary of DBMS_CAPTURE_ADM Subprograms

Table 8–1 DBMS_CAPTURE_ADM Subprograms

Subprogram	Description
"ABORT_GLOBAL_INSTANTIATION Procedure" on page 8-3	Reverses the effects of running the PREPARE_GLOBAL_INSTANTIATION procedure
"ABORT_SCHEMA_INSTANTIATION Procedure" on page 8-3	Reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION procedure
"ABORT_TABLE_INSTANTIATION Procedure" on page 8-4	Reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure
"ALTER_CAPTURE Procedure" on page 8-4	Alters a capture process
"CREATE_CAPTURE Procedure" on page 8-6	Creates a capture process
"DROP_CAPTURE Procedure" on page 8-8	Drops a capture process
"PREPARE_GLOBAL_INSTANTIATION Procedure" on page 8-8	Performs the synchronization necessary for instantiating all the tables in the database at another database
"PREPARE_SCHEMA_INSTANTIATION Procedure" on page 8-9	Performs the synchronization necessary for instantiating all tables in the schema at another database
"PREPARE_TABLE_INSTANTIATION Procedure" on page 8-10	Performs the synchronization necessary for instantiating the table at another database
"SET_PARAMETER Procedure" on page 8-11	Sets a capture process parameter to the specified value
"START_CAPTURE Procedure" on page 8-14	Starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue
"STOP_CAPTURE Procedure" on page 8-15	Stops the capture process from mining redo logs

Note: All procedures commit unless specified otherwise.

ABORT_GLOBAL_INSTANTIATION Procedure

Reverses the effects of running the PREPARE_GLOBAL_INSTANTIATION procedure.

Specifically, running this procedure removes data dictionary information related to the database instantiation.

Syntax

```
DBMS_CAPTURE_ADM.ABORT_GLOBAL_INSTANTIATION();
```

ABORT_SCHEMA_INSTANTIATION Procedure

Reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION procedure.

Specifically, running this procedure removes data dictionary information related to the schema instantiation.

Syntax

```
DBMS_CAPTURE_ADM.ABORT_SCHEMA_INSTANTIATION(
   schema_name     IN   VARCHAR2);
```

Parameter

Table 8–2 ABORT_SCHEMA_INSTANTIATION Procedure Parameter

Parameter	Description
schema_name	The name of the schema for which to abort the effects of preparing instantiation.

ABORT_TABLE_INSTANTIATION Procedure

Reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure.

Specifically, running this procedure removes data dictionary information related to the table instantiation.

Syntax

```
DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(
   table_name     IN   VARCHAR2);
```

Parameter

Table 8–3 ABORT_TABLE_INSTANTIATION Procedure Parameter

Parameter	Description
table_name	The name of the table for which to abort the effects of preparing instantiation, specified as [*schema_name.*]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.

ALTER_CAPTURE Procedure

Alters a capture process.

Syntax

```
DBMS_CAPTURE_ADM.ALTER_CAPTURE(
   capture_name      IN VARCHAR2,
   rule_set_name     IN VARCHAR2  DEFAULT NULL,
   remove_rule_set   IN BOOLEAN   DEFAULT false,
   start_scn         IN NUMBER    DEFAULT NULL);
```

Parameters

Table 8–4 ALTER_CAPTURE Procedure Parameters

Parameter	Description
capture_name	The name of the capture process being altered. You must specify an existing capture process name.
rule_set_name	The name of the rule set that contains the capture rules for this capture process. If you want to use a rule set for the capture process, then you must specify an existing rule set in the form [*schema_name.*]*rule_set_name*. For example, to specify a rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_RULE_ADM package. **See Also:** *Oracle9i Streams* for more information about the changes that can be captured by a capture process
remove_rule_set	If true, then removes the rule set for the specified capture process. If you remove a rule set for a capture process, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If false, then retains any rule set for the specified capture process. If the rule_set_name parameter is non-NULL, then this parameter should be set to false.
start_scn	A valid past SCN for the database where the capture process is capturing changes. The capture process will capture changes starting at the SCN specified. The SCN value specified must be from a point-in-time after the first capture process was created for the database. The first capture process for the database may or may not be the capture process being altered. An error is returned if an invalid SCN is specified. **Note:** When you change the start SCN for a capture process, the capture process is stopped and restarted automatically.

CREATE_CAPTURE Procedure

Creates a capture process.

The user who runs the CREATE_CAPTURE procedure is the user who captures changes. This user must have the necessary privileges to capture changes. These privileges include the following:

- Execute privilege on the rule set used by the capture process
- Execute privilege on all transformation functions used in the rule set
- Enqueue privilege on the queue used by the capture process

Note: Creation of the first capture process in a database may take some time because the data dictionary is duplicated during this creation.

See Also: *Oracle9i Streams* and Chapter 64, "DBMS_RULE_ADM" for more information about rules and rule sets

Syntax

```
DBMS_CAPTURE_ADM.CREATE_CAPTURE(
   queue_name      IN VARCHAR2,
   capture_name    IN VARCHAR2,
   rule_set_name   IN VARCHAR2  DEFAULT NULL,
   start_scn       IN NUMBER    DEFAULT NULL);
```

Parameters

Table 8–5 CREATE_CAPTURE Procedure Parameters

Parameter	Description
queue_name	The name of the queue into which the capture process enqueues changes. You must specify an existing queue in the form [*schema_name.*]*queue_name*. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default. **Note:** The queue_name setting cannot be altered after the capture process is created.
capture_name	The name of the capture process being created. A NULL specification is not allowed. **Note:** The capture_name setting cannot be altered after the capture process is created.
rule_set_name	The name of the rule set that contains the capture rules for this capture process. If you want to use a rule set for the capture process, then you must specify an existing rule set in the form [*schema_name.*]*rule_set_name*. For example, to specify a rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_RULE_ADM package. If you specify NULL, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. **See Also:** *Oracle9i Streams* for more information about the changes that can be captured by a capture process
start_scn	A valid past SCN for the database where the capture process is capturing changes. The capture process will capture changes starting at the SCN specified. The SCN value specified must be from a point in time after the first capture process was created for the database. If the capture process being created is the first capture process ever created for the current database, then you must specify NULL for the start_scn. An error is returned if an invalid SCN is specified.

DROP_CAPTURE Procedure

Drops a capture process.

Syntax

```
DBMS_CAPTURE_ADM.DROP_CAPTURE(
   capture_name    IN VARCHAR2);
```

Parameter

Table 8–6 DROP_CAPTURE Procedure Parameter

Parameter	Description
capture_name	The name of the capture process being dropped. Specify an existing capture process name.

PREPARE_GLOBAL_INSTANTIATION Procedure

Performs the synchronization necessary for instantiating all the tables in the database at another database. Run this procedure at the source database.

This procedure records the lowest SCN of each object in the database for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the database for instantiation.

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION;
```

PREPARE_SCHEMA_INSTANTIATION Procedure

Performs the synchronization necessary for instantiating all tables in the schema at another database. Run this procedure at the source database.

This procedure records the lowest SCN of each object in the schema for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the schema for instantiation.

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
   schema_name  IN VARCHAR2);
```

Parameter

Table 8–7 PREPARE_SCHEMA_INSTANTIATION Procedure Parameter

Parameter	Description
schema_name	The name of the schema. For example, hr.

PREPARE_TABLE_INSTANTIATION Procedure

Performs the synchronization necessary for instantiating the table at another database. Run this procedure at the source database.

This procedure records the lowest SCN of the table for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object.

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
   table_name  IN VARCHAR2);
```

Parameters

Table 8–8 PREPARE_TABLE_INSTANTIATION Procedure Parameter

Parameter	Description
`table_name`	The name of the table specified as `[`*`schema_name`*`.]`*`object_name`*. For example, `hr.employees`. If the schema is not specified, then the current user is the default.

SET_PARAMETER Procedure

Sets a capture process parameter to the specified value.

When you alter a parameter value, a short amount of time may pass before the new value for the parameter takes effect.

Syntax

```
DBMS_CAPTURE_ADM.SET_PARAMETER(
   capture_name   IN VARCHAR2,
   parameter      IN VARCHAR2,
   value          IN VARCHAR2);
```

Parameters

Table 8–9 SET_PARAMETER Procedure Parameters

Parameter	Description
`capture_name`	The name of the capture process. The capture process uses LogMiner to capture changes from the redo logs.
`parameter`	The name of the parameter you are setting. See "Capture Process Parameters" on page 8-12 for a list of these parameters.
`value`	The value to which the parameter is set

Capture Process Parameters

The following table lists the parameters for the capture process.

Table 8–10 Capture Process Parameters **(Page 1 of 2)**

Parameter Name	Possible Values	Default	Description
disable_on_limit	Y or N	N	If Y, then the capture process is disabled if the capture process terminates because it reached a value specified by the time_limit parameter or message_limit parameter. If N, then the capture process is restarted immediately after stopping because it reached a limit.
maximum_scn	A valid SCN or infinite	infinite	The capture process is disabled before capturing a change record with an SCN greater than or equal to the value specified. If infinite, then the capture process runs regardless of the SCN value.
message_limit	A positive integer or infinite	infinite	The capture process stops after capturing the specified number of messages. If infinite, then the capture process continues to run regardless of the number of messages captured.
parallelism	A positive integer	1	The number of parallel execution servers that may concurrently mine the redo log **Note:** ▪ When you change the value of this parameter, the capture process is stopped and restarted automatically. ▪ Setting the parallelism parameter to a number higher than the number of available parallel execution servers may disable the capture process. Make sure the PROCESSES and PARALLEL_MAX_SERVERS initialization parameters are set appropriately when you set the parallelism capture process parameter.

Table 8–10 Capture Process Parameters **(Page 2 of 2)**

Parameter Name	Possible Values	Default	Description
startup_seconds	0, a positive integer, or infinite	0	The maximum number of seconds to wait for another instantiation of the same capture process to finish. If the other instantiation of the same capture process does not finish within this time, then the capture process does not start. If infinite, then a capture process does not start until another instantiation of the same capture process finishes.
time_limit	A positive integer or infinite	infinite	The capture process stops as soon as possible after the specified number of seconds since it started. If infinite, then the capture process continues to run until it is stopped explicitly.
trace_level	0 or a positive integer	0	Set this parameter only under the guidance of Oracle Support Services.
write_alert_log	Y or N	Y	If Y, then the capture process writes a message to the alert log on exit. If N, then the capture process does not write a message to the alert log on exit. The message specifies the reason the capture process stopped.

Note:

- For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify infinite for larger values.
- For parameters that require an SCN setting, any valid SCN value can be specified.

START_CAPTURE Procedure

Starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue.

The start status is persistently recorded. Hence, if the status is ENABLED, then the capture process is started upon database instance startup.

The capture process is a background Oracle process and is prefixed by CP.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the start status of a capture process.

You can create the capture process using the following procedures:

- DBMS_CAPTURE_ADM.CREATE_CAPTURE
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_TABLE_RULES

See Also: Chapter 73, "DBMS_STREAMS_ADM"

Syntax

```
DBMS_CAPTURE_ADM.START_CAPTURE(
   capture_name  IN VARCHAR2);
```

Parameter

Table 8–11 START_CAPTURE Procedure Parameter

Parameter	Description
capture_name	The name of the capture process. The capture process uses LogMiner to capture changes in the redo information. A NULL setting is not allowed.

STOP_CAPTURE Procedure

Stops the capture process from mining redo logs.

The stop status is persistently recorded. Hence, if the status is DISABLED, then the capture process is not started upon database instance startup.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the stop status of a capture process.

Syntax

```
DBMS_CAPTURE_ADM.STOP_CAPTURE(
   capture_name  IN VARCHAR2,
   force         IN BOOLEAN  DEFAULT false);
```

Parameters

Table 8–12 STOP_CAPTURE Procedure Parameters

Parameter	Description
capture_name	The name of the capture process. A NULL setting is not allowed.
force	If TRUE, then stops the capture process instantly. If FALSE, then stops the capture process after the capture process captures its current transaction.

9

DBMS_DDL

This package provides access to some SQL data definition language (DDL) statements from stored procedures. It also provides special administration operations that are not available as DDLs.

The `ALTER_COMPILE` and `ANALYZE_OBJECT` procedures commit the current transaction, perform the operation, and then commit again.

This package runs with the privileges of the calling user, rather than the package owner SYS.

This chapter discusses the following topics:

Summary of DBMS_DDL Subprograms

Table 9–1 DBMS_DDL Package Subprograms

Subprogram	Description
ALTER_COMPILE Procedure on page 9-2	Compiles the PL/SQL object.
ANALYZE_OBJECT Procedure on page 9-3	Provides statistics for the database object.
IS_TRIGGER_FIRE_ONCE Function on page 9-4	Returns TRUE if the specified DML or DDL trigger is set to fire once. Otherwise, returns FALSE.
SET_TRIGGER_FIRING_PROPERTY Procedure on page 9-5	Sets the specified DML or DDL trigger's firing property.
ALTER_TABLE_REFERENCEABLE Procedure on page 9-7	Reorganizes object tables and swizzles references
ALTER_TABLE_NOT_REFERENCEABLE Procedure on page 9-7	Reorganizes object tables and swizzles references

ALTER_COMPILE Procedure

This procedure is equivalent to the following SQL statement:

```
ALTER PROCEDURE|FUNCTION|PACKAGE [<schema>.] <name> COMPILE [BODY]
```

Syntax

```
DBMS_DDL.ALTER_COMPILE (
   type   VARCHAR2,
   schema VARCHAR2,
   name   VARCHAR2);
```

Parameters

Table 9–2 ALTER_COMPILE Procedure Parameters

Parameter	Description
type	Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER.
schema	Schema name. If NULL, then use current schema (case-sensitive).

Table 9–2 ALTER_COMPILE Procedure Parameters

Parameter	Description
name	Name of the object (case-sensitive).

Exceptions

Table 9–3 ALTER_COMPILE Procedure Exceptions

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist.
ORA-20001:	Remote object, cannot compile.
ORA-20002:	Bad value for object type Should be either PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, or TRIGGER.

ANALYZE_OBJECT Procedure

This procedure provides statistics for the given table, index, or cluster. It is equivalent to the following SQL statement:

```
ANALYZE TABLE|CLUSTER|INDEX [<schema>.]<name> [<method>] STATISTICS [SAMPLE <n>
[ROWS|PERCENT]]
```

Syntax

```
DBMS_DDL.ANALYZE_OBJECT (
   type             VARCHAR2,
   schema           VARCHAR2,
   name             VARCHAR2,
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL,
   partname         VARCHAR2 DEFAULT NULL);
```

Parameters

Table 9–4 ANALYZE_OBJECT Procedure Parameters

Parameter	Description
type	One of TABLE, CLUSTER or INDEX. If none of these, an ORA-20001 error is raised.
schema	Schema of object to analyze. NULL means current schema, case-sensitive.
name	Name of object to analyze, case-sensitive.
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format. [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]
partname	Specific partition to be analyzed.

Exceptions

Table 9–5 ANALYZE_OBJECT Procedure Exceptions

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist.
ORA-20001:	Bad value for object type. Should be either TABLE, INDEX or CLUSTER.
ORA-20002:	METHOD must be one of COMPUTE, ESTIMATE or DELETE.

IS_TRIGGER_FIRE_ONCE Function

This function returns TRUE if the specified DML or DDL trigger is set to fire once. Otherwise, it returns FALSE.

A fire once trigger fires in a user session but does not fire in the following cases:

- For changes made by a Streams apply process
- For changes made by executing one or more Streams apply errors using the EXECUTE_ERROR or EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package

Note: Only DML and DDL triggers can be fire once. All other types of triggers always fire.

See Also: "SET_TRIGGER_FIRING_PROPERTY Procedure" on page 9-5

Syntax

```
DBMS_DDL.IS_TRIGGER_FIRE_ONCE
   trig_owner          IN  VARCHAR2,
   trig_name           IN  VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 9–6 IS_TRIGGER_FIRE_ONCE Function Parameters

Parameter	Description
trig_owner	Schema of trigger
trig_name	Name of trigger

SET_TRIGGER_FIRING_PROPERTY Procedure

This procedure sets the specified DML or DDL trigger's firing property. Use this procedure to control a DML or DDL trigger's firing property for changes:

- Applied by a Streams apply process
- Made by executing one or more Streams apply errors using the EXECUTE_ERROR or EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package.

You can specify one of the following settings for a trigger's firing property:

- If the `fire_once` parameter is set to `TRUE` for a trigger, then the trigger does not fire for these types of changes.
- If the `fire_once` parameter is set to `FALSE` for a trigger, then the trigger fires for these types of changes.

Regardless of the firing property set by this procedure, a trigger continues to fire when changes are made by means other than the apply process or apply error execution. For example, if a user session or an application makes a change, then the trigger continues to fire, regardless of the firing property.

Note:

- If you dequeue an error transaction from the error queue and execute it without using the `DBMS_APPLY_ADM` package, then relevant changes resulting from this execution cause a trigger to fire, regardless of the trigger firing property.
- Only DML and DDL triggers can be fire once. All other types of triggers always fire.

See Also: *Oracle9i Streams* for more information about the apply process and controlling a trigger's firing property

Syntax

```
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY
   trig_owner         IN  VARCHAR2,
   trig_name          IN  VARCHAR2,
   fire_once          IN  BOOLEAN);
```

Parameters

Table 9–7 SET_TRIGGER_FIRING_PROPERTY Procedure Parameters

Parameter	Description
`trig_owner`	Schema of the trigger to set
`trig_name`	Name of the trigger to set
`fire_once`	If `TRUE`, then the trigger is set to fire once. By default, the `fire_once` parameter is set to `TRUE` for DML and DDL triggers. If `FALSE`, then the trigger is set to always fire.

ALTER_TABLE_REFERENCEABLE Procedure

This procedure reorganizes object tables and swizzles references. For example, assume you have an object table FOO and that references in other tables point to objects stored in FOO. If you want to change some of the table organization—for example, you want to make it an IOT or a partitioned table, or you want to reorganize the data more efficiently—you copy all data from FOO into FOO2. Then you use the `alter_table_referenceable` and `alter_table_not_referenceable` procedures to swizzle all existing references to point to FOO2 instead of FOO.

Syntax

```
DBMS_DDL.ALTER_TABLE_REFERENCEABLE
   TABLE_NAME        IN           VARCHAR2,
   TABLE_SCHEMA      IN  DEFAULT  VARCHAR2,
   AFFECTED_SCHEMA   IN  DEFAULT  VARCHAR2;
```

ALTER_TABLE_NOT_REFERENCEABLE Procedure

See ALTER_TABLE_NOT_REFERENCEABLE Procedure on page 9-7.

Syntax

```
DBMS_DDL.ALTER_TABLE_NOT_REFERENCEABLE
   TABLE_NAME        IN           VARCHAR2,
   TABLE_SCHEMA      IN  DEFAULT  VARCHAR2,
   AFFECTED_SCHEMA   IN  DEFAULT  VARCHAR2;
```

10

DBMS_DEBUG

DBMS_DEBUG is a PL/SQL API to the PL/SQL debugger layer, Probe, in the Oracle server.

This API is primarily intended to implement server-side debuggers and it provides a way to debug server-side PL/SQL program units.

> **Note:** The term *program unit* refers to a PL/SQL program of any type (procedure, function, package, package body, trigger, anonymous block, object type, or object type body).

This chapter discusses the following topics:

Using DBMS_DEBUG

To debug server-side code, you must have two database sessions: one session to run the code in debug mode (the target session), and a second session to supervise the target session (the debug session).

The target session becomes available for debugging by making initializing calls with DBMS_DEBUG. This marks the session so that the PL/SQL interpreter runs in debug mode and generates debug events. As debug events are generated, they are posted from the session. In most cases, debug events require return notification: the interpreter pauses awaiting a reply.

Meanwhile, the debug session must also initialize itself using DBMS_DEBUG: This tells it which target session to supervise. The debug session may then call entry points in DBMS_DEBUG to read events that were posted from the target session and to communicate with the target session.

DBMS_DEBUG does not provide an interface to the PL/SQL compiler; however, it does depend on debug information optionally generated by the compiler. Without debug information, it is not possible to examine or modify the values of parameters or variables. There are two ways to ensure that debug information is generated: through a session switch, or through individual recompilation.

To set the session switch, enter the following statement:

```
ALTER SESSION SET PLSQL_DEBUG = true;
```

This instructs the compiler to generate debug information for the remainder of the session. It does not recompile any existing PL/SQL.

To generate debug information for existing PL/SQL code, use one of the following statements (the second recompiles a package or type body):

```
ALTER [PROCEDURE | FUNCTION | PACKAGE | TRIGGER | TYPE] <name> COMPILE DEBUG;
ALTER [PACKAGE | TYPE] <name> COMPILE DEBUG BODY;
```

Figure 10–1 and Figure 10–2 illustrate the flow of operations in the session to be debugged and in the debugging session.

Figure 10–1 Target Session

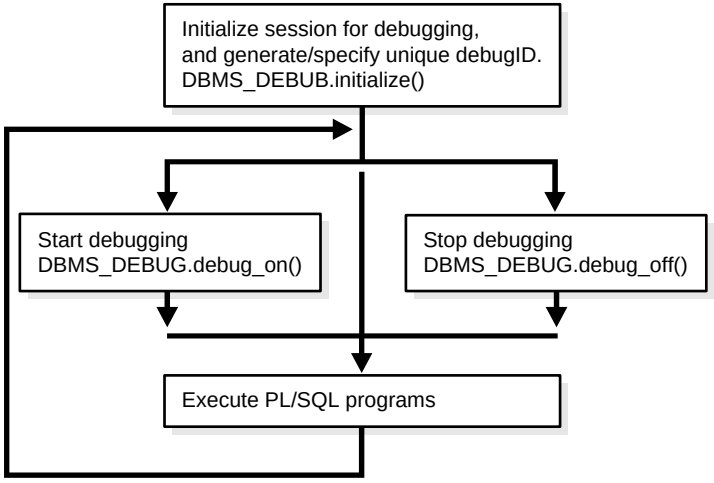


Figure 10–2 Debug Session

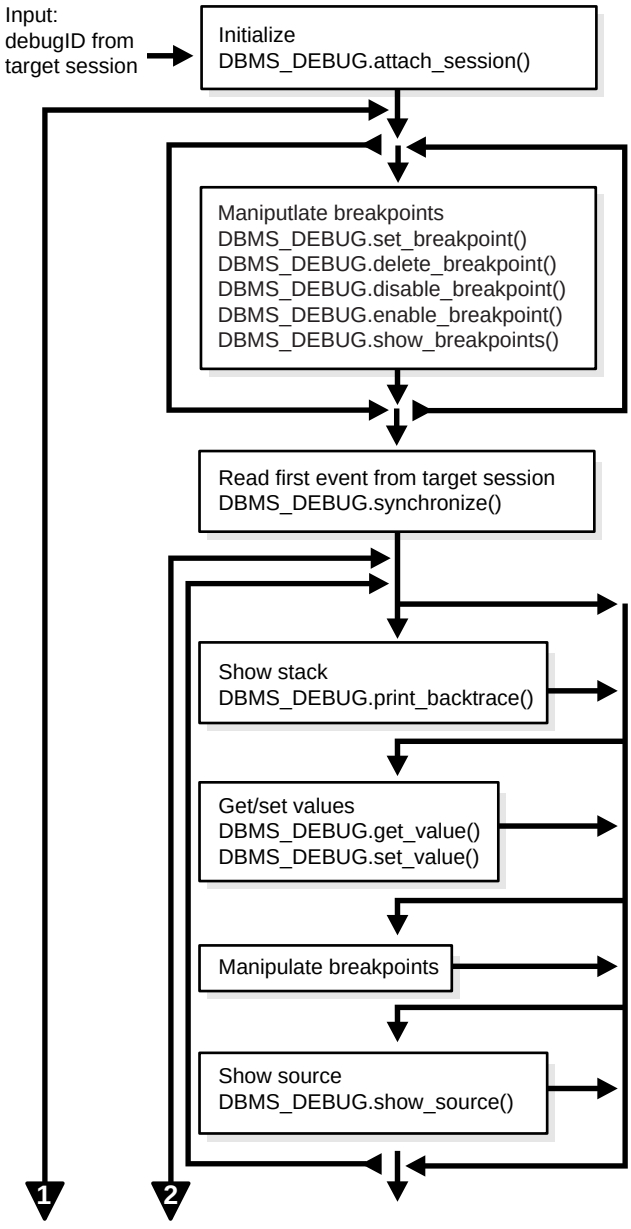
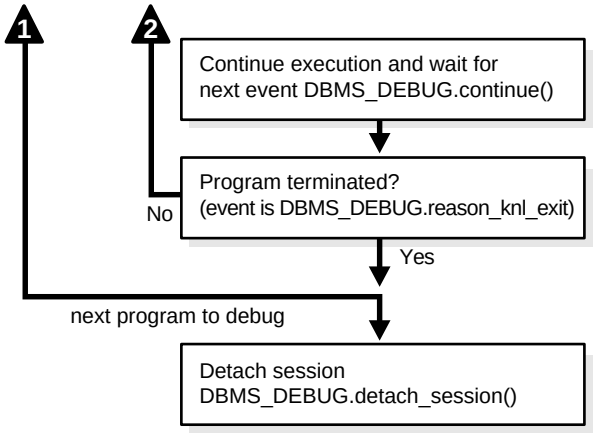


Figure 10–2 Debug Session (Cont.)



Control of the Interpreter

The interpreter pauses execution at the following times:

1. At startup of the interpreter so any deferred breakpoints may be installed prior to execution.
2. At any line containing an enabled breakpoint.
3. At any line where an *interesting* event occurs. The set of interesting events is specified by the flags passed to `DBMS_DEBUG.CONTINUE` in the `breakflags` parameter.

Usage Notes

Session Termination

There is no event for session termination. Therefore, it is the responsibility of the debug session to check and make sure that the target session has not ended. A call to `DBMS_DEBUG.SYNCHRONIZE` after the target session has ended causes the debug session to hang until it times out.

Deferred Operations

The diagram suggests that it is possible to set breakpoints prior to having a target session. This is true. In this case, Probe caches the breakpoint request and transmits

it to the target session at first synchronization. However, if a breakpoint request is deferred in this fashion, then:

- SET_BREAKPOINT does not set the breakpoint number (it can be obtained later from SHOW_BREAKPOINTS if necessary).
- SET_BREAKPOINT does not validate the breakpoint request. If the requested source line does not exist, then an error silently occurs at synchronization, and no breakpoint is set.

Diagnostic Output

To debug Probe, there are *diagnostics* parameters to some of the calls in DBMS_DEBUG. These parameters specify whether to place diagnostic output in the RDBMS tracefile. If output to the RDBMS tracefile is disabled, these parameters have no effect.

Types and Constants

PROGRAM_INFO Types

This type specifies a program location. It is a line number in a program unit. This is used for stack backtraces and for setting and examining breakpoints. The read-only fields are currently ignored by Probe for breakpoint operations. They are set by Probe only for stack backtraces.

Type	Description
EntrypointName	Null, unless this is a nested procedure or function.
LibunitType	Disambiguate among objects that share the same namespace (for example, procedure and package specifications). See the Libunit Types on page 10-9 for more information.

```
TYPE program_info IS RECORD
(
    -- The following fields are used when setting a breakpoint
    Namespace        BINARY_INTEGER,  -- See 'NAMESPACES' section below.
    Name             VARCHAR2(30),    -- name of the program unit
    Owner            VARCHAR2(30),    -- owner of the program unit
    Dblink           VARCHAR2(30),    -- database link, if remote
    Line#            BINARY_INTEGER,
    -- Read-only fields (set by Probe when doing a stack backtrace)
    LibunitType      BINARY_INTEGER,
    EntrypointName   VARCHAR2(30)
);
```

RUNTIME_INFO Type

This type gives context information about the running program.

```
TYPE runtime_info IS RECORD
(
    Line#            BINARY_INTEGER,  -- (duplicate of program.line#)
    Terminated       BINARY_INTEGER,  -- has the program terminated?
    Breakpoint       BINARY_INTEGER,  -- breakpoint number
    StackDepth       BINARY_INTEGER,  -- number of frames on the stack
    InterpreterDepth BINARY_INTEGER,  -- <reserved field>
    Reason           BINARY_INTEGER,  -- reason for suspension
    Program          program_info     -- source location
);
```

BREAKPOINT_INFO Type

This type gives information about a breakpoint, such as its current status and the program unit in which it was placed.

```
TYPE breakpoint_info IS RECORD
(
   -- These fields are duplicates of 'program_info':
   Name        VARCHAR2(30),
   Owner       VARCHAR2(30),
   DbLink      VARCHAR2(30),
   Line#       BINARY_INTEGER,
   LibunitType BINARY_INTEGER,
   Status      BINARY_INTEGER   -- see breakpoint_status_* below
);
```

INDEX_TABLE Type

This type is used by GET_INDEXES to return the available indexes for an indexed table.

```
TYPE index_table IS table of BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

BACKTRACE_TABLE Type

This type is used by PRINT_BACKTRACE.

```
TYPE backtrace_table IS TABLE OF program_info INDEX BY BINARY_INTEGER;
```

BREAKPOINT_TABLE Type

This type is used by SHOW_BREAKPOINTS.

```
TYPE breakpoint_table IS TABLE OF breakpoint_info INDEX BY BINARY_INTEGER;
```

VC2_TABLE Type

This type is used by SHOW_SOURCE.

```
TYPE vc2_table IS TABLE OF VARCHAR2(90) INDEX BY BINARY_INTEGER;
```

Constants

A breakpoint status may have the following value:

- `breakpoint_status_unused`—breakpoint is not in use

Otherwise, the status is a mask of the following values:

- `breakpoint_status_active`—a line breakpoint
- `breakpoint_status_disabled`—breakpoint is currently disabled
- `breakpoint_status_remote`—a shadow breakpoint (a local representation of a remote breakpoint)

NAMESPACES

Program units on the server reside in different namespaces. When setting a breakpoint, specify the desired namespace.

1. `Namespace_cursor` contains cursors (anonymous blocks).
2. `Namespace_pgkspec_or_toplevel` contains:
   - Package specifications.

- Procedures and functions that are not nested inside other packages, procedures, or functions.
- Object types.

3. `Namespace_pkg_body` contains package bodies and type bodies.
4. `Namespace_trigger` contains triggers.

Libunit Types

These values are used to disambiguate among objects in a given namespace. These constants are used in `PROGRAM_INFO` when Probe is giving a stack backtrace.

- `LibunitType_cursor`
- `LibunitType_procedure`
- `LibunitType_function`
- `LibunitType_package`
- `LibunitType_package_body`
- `LibunitType_trigger`
- `LibunitType_Unknown`

Breakflags

These are values to use for the `breakflags` parameter to `CONTINUE`, in order to tell Probe what events are of interest to the client. These flags may be combined.

Value	Description
`break_next_line`	Break at next source line (step over calls).
`break_any_call`	Break at next source line (step into calls).
`break_any_return`	Break after returning from current entrypoint (skip over any entrypoints called from the current routine).
`break_return`	Break the next time an entrypoint gets ready to return. (This includes entrypoints called from the current one. If interpreter is running `Proc1`, which calls `Proc2`, then `break_return` stops at the end of `Proc2`.)
`break_exception`	Break when an exception is raised.
`break_handler`	Break when an exception handler is executed.

Value	Description
abort_execution	Stop execution and force an 'exit' event as soon as DBMS_DEBUG.CONTINUE is called.

Information Flags

These are flags which may be passed as the info_requested parameter to SYNCHRONIZE, CONTINUE, and GET_RUNTIME_INFO.

Flag	Description
info_getStackDepth	Get the current depth of the stack.
info_getBreakpoint	Get the breakpoint number.
info_getLineinfo	Get program unit information.

Reasons for Suspension

After CONTINUE is run, the program either runs to completion or breaks on some line.

Reason	Description
reason_none	-
reason_interpreter_starting	Interpreter is starting.
reason_breakpoint	Hit a breakpoint.
reason_enter	Procedure entry.
reason_return	Procedure is about to return.
reason_finish	Procedure is finished.
reason_line	Reached a new line.
reason_interrupt	An interrupt occurred.
reason_exception	An exception was raised.
reason_exit	Interpreter is exiting (old form).
reason_knl_exit	Kernel is exiting.
reason_handler	Start exception-handler.

Reason	Description
reason_timeout	A timeout occurred.
reason_instantiate	Instantiation block.
reason_abort	Interpreter is aborting.

Error Codes, Exceptions, and Variables

Error Codes

These values are returned by the various functions called in the debug session (SYNCHRONIZE, CONTINUE, SET_BREAKPOINT, and so on). If PL/SQL exceptions worked across client/server and server/server boundaries, then these would all be exceptions rather than error codes.

Value	Description
success	Normal termination.

Statuses returned by GET_VALUE and SET_VALUE:

Status	Description
error_bogus_frame	No such entrypoint on the stack.
error_no_debug_info	Program was compiled without debug symbols.
error_no_such_object	No such variable or parameter.
error_unknown_type	Debug information is unreadable.
error_indexed_table	Returned by GET_VALUE if the object is a table, but no index was provided.
error_illegal_index	No such element exists in the collection.
error_nullcollection	Table is atomically null.
error_nullvalue	Value is null.

Statuses returned by SET_VALUE:

Status	Description
error_illegal_value	Constraint violation.
error_illegal_null	Constraint violation.
error_value_malformed	Unable to decipher the given value.
error_other	Some other error.
error_name_incomplete	Name did not resolve to a scalar.

Statuses returned by the breakpoint functions:

Status	Description
error_no_such_breakpt	No such breakpoint.
error_idle_breakpt	Cannot enable or disable an unused breakpoint.
error_bad_handle	Unable to set breakpoint in given program (nonexistent or security violation).

General error codes (returned by many of the DBMS_DEBUG subprograms):

Status	Description
error_unimplemented	Functionality is not yet implemented.
error_deferred	No program running; operation deferred.
error_exception	An exception was raised in the DBMS_DEBUG or Probe packages on the server.
error_communication	Some error other than a timeout occurred.
error_timeout	Timout occurred.

Exceptions

Exception	Description
illegal_init	DEBUG_ON was called prior to INITIALIZE.

The following exceptions are raised by procedure SELF_CHECK:

Exception	Description
pipe_creation_failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_mismatch	Datatype in the pipe was wrong.
pipe_data_error	Data got garbled in the pipe.

Variables

Exception	Description
default_timeout	The timeout value (used by both sessions).The smallest possible timeout is 1 second. If this value is set to 0, then a large value (3600) is used.

Common and Debug Session Sections

Common Section

The following subprograms may be called in either the target or the debug session:

- PROBE_VERSION Procedure
- SELF_CHECK Procedure
- SET_TIMEOUT Function

Debug Session Section

The following subprograms should be run in the debug session only:

- ATTACH_SESSION Procedure
- SYNCHRONIZE Function
- SHOW_SOURCE Procedure
- PRINT_BACKTRACE Procedure
- CONTINUE Function
- SET_BREAKPOINT Function
- DELETE_BREAKPOINT Function
- DISABLE_BREAKPOINT Function
- ENABLE_BREAKPOINT Function
- SHOW_BREAKPOINTS Procedure
- GET_VALUE Function
- SET_VALUE Function
- DETACH_SESSION Procedure
- GET_RUNTIME_INFO Function
- GET_INDEXES Function
- EXECUTE Procedure

OER Breakpoints

Exceptions that are declared in PL/SQL programs are known as user-defined exceptions. In addition, there are Oracle Errors (OERs) that are returned from the Oracle kernel. To tie the two mechanisms together, PL/SQL provides the `exception_init` pragma that turns a user-defined exception into an OER, so that a PL/SQL handler may be used for it, and so that the PL/SQL engine can return OERs to the Oracle kernel. As of the current release, the only information available about an OER is its number. If two user-defined exceptions are exception_init'd to the same OER, they are indistinguishable.

Summary of DBMS_DEBUG Subprograms

Table 10–1 DBMS_DEBUG Package Subprograms

Subprogram	Description
PROBE_VERSION Procedure on page 10-16	Returns the version number of DBMS_DEBUG on the server.
SELF_CHECK Procedure on page 10-16	Performs an internal consistency check.
SET_TIMEOUT Function on page 10-17	Sets the timeout value.
INITIALIZE Function on page 10-18	Sets debugID in target session.
DEBUG_ON Procedure on page 10-19	Turns debug-mode on.
DEBUG_OFF Procedure on page 20	Turns debug-mode off.
ATTACH_SESSION Procedure on page 10-20	Notifies the debug session about the target debugID.
SYNCHRONIZE Function on page 10-21	Waits for program to start running.
SHOW_SOURCE Procedure on page 10-22	Fetches program source.
PRINT_BACKTRACE Procedure on page 10-24	Prints a stack backtrace.
CONTINUE Function on page 10-24	Continues execution of the target program.
SET_BREAKPOINT Function on page 10-25	Sets a breakpoint in a program unit.
DELETE_BREAKPOINT Function on page 10-27	Deletes a breakpoint.
DISABLE_BREAKPOINT Function on page 10-27	Disables a breakpoint.
ENABLE_BREAKPOINT Function on page 10-28	Activates an existing breakpoint.
SHOW_BREAKPOINTS Procedure on page 10-29	Returns a listing of the current breakpoints.

Table 10–1 (Cont.) DBMS_DEBUG Package Subprograms

Subprogram	Description
GET_VALUE Function on page 10-30	Gets a value from the currently-running program.
SET_VALUE Function on page 10-32	Sets a value in the currently-running program.
DETACH_SESSION Procedure on page 10-34	Stops debugging the target program.
GET_RUNTIME_INFO Function on page 10-34	Returns information about the current program.
GET_INDEXES Function on page 10-35	Returns the set of indexes for an indexed table.
EXECUTE Procedure on page 10-36	Executes SQL or PL / SQL in the target session.

PROBE_VERSION Procedure

This procedure returns the version number of DBMS_DEBUG on the server.

Syntax

```
DBMS_DEBUG.PROBE_VERSION (
   major out BINARY_INTEGER,
   minor out BINARY_INTEGER);
```

Parameters

Table 10–2 PROBE_VERSION Procedure Parameters

Parameter	Description
major	Major version number.
minor	Minor version number: increments as functionality is added.

SELF_CHECK Procedure

This procedure performs an internal consistency check. SELF_CHECK also runs a communications test to ensure that the Probe processes are able to communicate.

If SELF_CHECK does not return successfully, then an incorrect version of DBMS_DEBUG was probably installed on this server. The solution is to install the correct version (pbload.sql loads DBMS_DEBUG and the other relevant packages).

Syntax

```
DBMS_DEBUG.SELF_CHECK (
   timeout IN binary_integer := 60);
```

Parameters

Table 10–3 SELF_CHECK Procedure Parameters

Parameter	Description
timeout	The timeout to use for the communication test. Default is 60 seconds.

Exceptions

Table 10–4 SELF_CHECK Procedure Exceptions

Exception	Description
OER-6516	Probe version is inconsistent.
pipe_creation_ failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_ mismatch	Datatype in the pipe was wrong.
pipe_data_error	Data got garbled in the pipe.

All of these exceptions are fatal. They indicate a serious problem with Probe that prevents it from working correctly.

SET_TIMEOUT Function

This function sets the timeout value and returns the new timeout value.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT (
   timeout BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–5 SET_TIMEOUT Function Parameters

Parameter	Description
`timeout`	The timeout to use for communication between the target and debug sessions.

TARGET SESSION Section

The following subprograms are run in the target session (the session that is to be debugged):

- INITIALIZE Function
- DEBUG_ON Procedure
- DEBUG_OFF Procedure

INITIALIZE Function

This function initializes the target session for debugging.

Syntax

```
DBMS_DEBUG.INITIALIZE (
   debug_session_id  IN VARCHAR2       := NULL,
   diagnostics       IN BINARY_INTEGER := 0)
  RETURN VARCHAR2;
```

Parameters

Table 10–6 INITIALIZE Function Parameters

Parameter	Description
`debug_session_id`	Name of session ID. If `NULL`, then a unique ID is generated.

Table 10–6 INITIALIZE Function Parameters

Parameter	Description
diagnostics	Indicates whether to dump diagnostic output to the tracefile. 0 = (default) no diagnostics 1 = print diagnostics

Returns

The newly-registered debug session ID (debugID)

DEBUG_ON Procedure

This procedure marks the target session so that all PL/SQL is run in debug mode. This must be done before any debugging can take place.

Syntax

```
DBMS_DEBUG.DEBUG_ON (
   no_client_side_plsql_engine BOOLEAN := TRUE,
   immediate                   BOOLEAN := FALSE);
```

Parameters

Table 10–7 DEBUG_ON Procedure Parameters

Parameter	Description
no_client_side_ plsql_engine	Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine.
immediate	If this is TRUE, then the interpreter immediately switches itself into debug-mode, instead of continuing in regular mode for the duration of the call.

Caution: There must be a debug session waiting if immediate is TRUE.

DEBUG_OFF Procedure

This procedure notifies the target session that debugging should no longer take place in that session. It is not necessary to call this function before ending the session.

Syntax

```
DBMS_DEBUG.DEBUG_OFF;
```

Usage Notes

The server does not handle this entrypoint specially. Therefore, it attempts to debug this entrypoint.

ATTACH_SESSION Procedure

This procedure notifies the debug session about the target program.

Syntax

```
DBMS_DEBUG.ATTACH_SESSION (
   debug_session_id  IN VARCHAR2,
   diagnostics       IN BINARY_INTEGER := 0);
```

Parameters

Table 10–8 ATTACH_SESSION Procedure Parameters

Parameter	Description
debug_session_id	Debug ID from a call to INITIALIZE in target session.
diagnostics	Generate diagnostic output if nonzero.

SYNCHRONIZE Function

This function waits until the target program signals an event. If `info_requested` is not `NULL`, then it calls `GET_RUNTIME_INFO`.

Syntax

```
DBMS_DEBUG.SYNCHRONIZE (
   run_info       OUT  runtime_info,
   info_requested IN   BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–9 SYNCHRONIZE Function Parameters

Parameter	Description
`run_info`	Structure in which to write information about the program. By default, this includes information about what program is running and at which line execution has paused.
`info_requested`	Optional bit-field in which to request information other than the default (which is `info_getStackDepth` + `info_getLineInfo`). 0 means that no information is requested at all. See "Information Flags" on page 10-10.

Returns

Table 10–10 SYNCHRONIZE Function Returns

Return	Description
`success`	
`error_timeout`	Timed out before the program started execution.
`error_communication`	Other communication error.

SHOW_SOURCE Procedure

The best way to get the source code (for a program that is being run) is to use SQL. For example:

```
DECLARE
   info DBMS_DEBUG.runtime_info;
BEGIN
   -- call DBMS_DEBUG.SYNCHRONIZE, CONTINUE,
   -- or GET_RUNTIME_INFO to fill in 'info'
   SELECT text INTO <buffer> FROM all_source
   WHERE owner = info.Program.Owner
     AND name  = info.Program.Name
     AND line  = info.Line#;
END;
```

However, this does not work for nonpersistent programs (for example, anonymous blocks and trigger invocation blocks). For nonpersistent programs, call SHOW_SOURCE. There are two flavors: one returns an indexed table of source lines, and the other returns a packed (and formatted) buffer.

There are two overloaded SHOW_SOURCE procedures.

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
   first_line  IN   BINARY_INTEGER,
   last_line   IN   BINARY_INTEGER,
   source      OUT  vc2_table);
```

Parameters

Table 10–11 SHOW_SOURCE Procedure Parameters

Parameter	Description
first_line	Line number of first line to fetch. (PL/SQL programs always start at line 1 and have no holes.)
last_line	Line number of last line to fetch. No lines are fetched past the end of the program.
source	The resulting table, which may be indexed by line#.

Returns

An indexed table of source-lines. The source lines are stored starting at `first_line`. If any error occurs, then the table is empty.

Usage Notes

This second overloading of `SHOW_SOURCE` returns the source in a formatted buffer, complete with line-numbers. It is faster than the indexed table version, but it does not guarantee to fetch all the source.

If the source does not fit in bufferlength (`buflen`), then additional pieces can be retrieved using the `GET_MORE_SOURCE` procedure (`pieces` returns the number of additional pieces that need to be retrieved).

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
   first_line   IN     BINARY_INTEGER,
   last_line    IN     BINARY_INTEGER,
   window       IN     BINARY_INTEGER,
   print_arrow  IN     BINARY_INTEGER,
   buffer       IN OUT VARCHAR2,
   buflen       IN     BINARY_INTEGER,
   pieces       OUT    BINARY_INTEGER);
```

Parameters

Table 10–12 SHOW_SOURCE Procedure Parameters

Parameter	Description
`first_line`	Smallest line-number to print.
`last_line`	Largest line-number to print.
`window`	'Window' of lines (the number of lines around the current source line).
`print_arrow`	Nonzero means to print an arrow before the current line.
`buffer`	Buffer in which to place the source listing.
`buflen`	Length of buffer.
`pieces`	Set to nonzero if not all the source could be placed into the given buffer.

PRINT_BACKTRACE Procedure

This procedure prints a backtrace listing of the current execution stack. This should only be called if a program is currently running.

There are two overloaded PRINT_BACKTRACE procedures.

Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (
  listing IN OUT VARCHAR2);
```

Parameters

Table 10–13 PRINT_BACKTRACE Procedure Parameters

Parameter	Description
listing	A formatted character buffer with embedded newlines.

Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (
   backtrace OUT backtrace_table);
```

Parameters

Table 10–14 PRINT_BACKTRACE Procedure Parameters

Parameter	Description
backtrace	1-based indexed table of backtrace entries. The currently-running procedure is the last entry in the table (that is, the frame numbering is the same as that used by GET_VALUE). Entry 1 is the oldest procedure on the stack.

CONTINUE Function

This function passes the given breakflags (a mask of the events that are of interest) to Probe in the target process. It tells Probe to continue execution of the target process, and it waits until the target process runs to completion or signals an event.

If info_requested is not NULL, then calls GET_RUNTIME_INFO.

Syntax

```
DBMS_DEBUG.CONTINUE (
   run_info       IN OUT runtime_info,
   breakflags     IN     BINARY_INTEGER,
   info_requested IN     BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–15 CONTINUE Function Parameters

Parameter	Description
`run_info`	Information about the state of the program.
`breakflags`	Mask of events that are of interest. See "Breakflags" on page 10-9.
`info_requested`	Which information should be returned in `run_info` when the program stops. See "Information Flags" on page 10-10.

Returns

Table 10–16 CONTINUE Function Returns

Return	Description
`success`	
`error_timeout`	Timed out before the program started running.
`error_communication`	Other communication error.

SET_BREAKPOINT Function

This function sets a breakpoint in a program unit, which persists for the current session. Execution pauses if the target program reaches the breakpoint.

Syntax

```
DBMS_DEBUG.SET_BREAKPOINT (
   program     IN  program_info,
   line#       IN  BINARY_INTEGER,
   breakpoint# OUT BINARY_INTEGER,
   fuzzy       IN  BINARY_INTEGER := 0,
```

```
 iterations  IN  BINARY_INTEGER := 0)
RETURN BINARY_INTEGER;
```

Parameters

Table 10–17 SET_BREAKPOINT Function Parameters

Parameter	Description
program	Information about the program unit in which the breakpoint is to be set. (In version 2.1 and later, the namespace, name, owner, and dblink may be set to NULL, in which case the breakpoint is placed in the currently-running program unit.)
line#	Line at which the breakpoint is to be set.
breakpoint#	On successful completion, contains the unique breakpoint number by which to refer to the breakpoint.
fuzzy	Only applicable if there is no executable code at the specified line: 0 means return error_illegal_line. 1 means search forward for an adjacent line at which to place the breakpoint. -1 means search backward for an adjacent line at which to place the breakpoint.
iterations	Number of times to wait before signalling this breakpoint.

Note: The fuzzy and iterations parameters are not yet implemented.

Returns

Table 10–18 SET_BREAKPOINT Function Returns

Return	Description
success	
error_illegal_line	Cannot set a breakpoint at that line.
error_bad_handle	No such program unit exists.

DELETE_BREAKPOINT Function

This function deletes a breakpoint.

Syntax

```
DBMS_DEBUG.DELETE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–19 DELETE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 10–20 DELETE_BREAKPOINT Function Returns

Return	Description
success	
error_no_such_ breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot delete an unused breakpoint.
error_stale_breakpt	The program unit was redefined since the breakpoint was set.

DISABLE_BREAKPOINT Function

This function makes an existing breakpoint inactive, but it leaves it in place.

Syntax

```
DBMS_DEBUG.DISABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–21 DISABLE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 10–22 DISABLE_BREAKPOINT Function Returns

Returns	Description
success	
error_no_such_ breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot disable an unused breakpoint.

ENABLE_BREAKPOINT Function

This function is the reverse of disabling. This enables a previously disabled breakpoint.

Syntax

```
DBMS_DEBUG.ENABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–23 ENABLE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 10–24 ENABLE_BREAKPOINT Function Returns

Return	Description
success	
error_no_such_breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot enable an unused breakpoint.

SHOW_BREAKPOINTS Procedure

This procedure returns a listing of the current breakpoints. There are two overloaded SHOW_BREAKPOINTS procedures.

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing    IN OUT VARCHAR2);
```

Parameters

Table 10–25 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
listing	A formatted buffer (including newlines) of the breakpoints.

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing  OUT breakpoint_table);
```

Parameters

Table 10–26 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
listing	Indexed table of breakpoint entries. The breakpoint number is indicated by the index into the table. Breakpoint numbers start at 1 and are reused when deleted.

GET_VALUE Function

This function gets a value from the currently-running program. There are two overloaded GET_VALUE functions.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   frame#         IN  BINARY_INTEGER,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 10–27 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
frame#	Frame in which it lives; 0 means the current procedure.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Returns

Table 10–28 GET_VALUE Function Returns

Return	Description
success	
error_bogus_frame	Frame does not exist.
error_no_debug_info	Entrypoint has no debug information.
error_no_such_object	variable_name does not exist in frame#.
error_unknown_type	The type information in the debug information is illegible.
error_nullvalue	Value is NULL.
error_indexed_table	The object is a table, but no index was provided.

This form of GET_VALUE is for fetching package variables. Instead of a frame#, it takes a handle, which describes the package containing the variable.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   handle         IN  program_info,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 10–29 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
handle	Description of the package containing the variable.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Returns

Table 10–30 GET_VALUE Function Returns

Return	Description
error_no_such_object	Either: - Package does not exist. - Package is not instantiated. - User does not have privileges to debug the package. - Object does not exist in the package.
error_indexed_table	The object is a table, but no index was provided.

Example

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```
DECLARE
```

```
   handle     dbms_debug.program_info;
   resultbuf  VARCHAR2(500);
   retval     BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.get_value('VAR', handle, resultbuf, NULL);
END;
```

SET_VALUE Function

This function sets a value in the currently-running program. There are two overloaded SET_VALUE functions.

Syntax

```
DBMS_DEBUG.SET_VALUE (
   frame#              IN binary_integer,
   assignment_statement IN varchar2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–31 SET_VALUE Function Parameters

Parameter	Description
frame#	Frame in which the value is to be set; 0 means the currently executing frame.
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

Returns

Table 10–32 SET_VALUE Function Returns

Return	Description
success	-

Table 10–32 SET_VALUE Function Returns

Return	Description
error_illegal_value	Not possible to set it to that value.
error_illegal_null	Cannot set to NULL because object type specifies it as 'not null'.
error_value_ malformed	Value is not a scalar.
error_name_ incomplete	The assignment statement does not resolve to a scalar. For example, 'x := 3;', if x is a record.

This form of SET_VALUE sets the value of a package variable.

Syntax

```
DBMS_DEBUG.SET_VALUE (
   handle              IN program_info,
   assignment_statement IN VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–33 SET_VALUE Function Parameters

Parameter	Description
handle	Description of the package containing the variable.
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

Table 10–34 SET_VALUE Function Returns

Return	Description
error_no_such_object	Either: - Package does not exist. - Package is not instantiated. - User does not have privileges to debug the package. - Object does not exist in the package.

In some cases, the PL/SQL compiler uses temporaries to access package variables, and Probe does not guarantee to update such temporaries. It is possible, although unlikely, that modification to a package variable using SET_VALUE might not take effect for a line or two.

Example

To set the value of SCOTT.PACK.var to 6:

```
DECLARE
   handle  dbms_debug.program_info;
   retval  BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.set_value(handle, 'var := 6;');
END;
```

DETACH_SESSION Procedure

This procedure stops debugging the target program. This procedure may be called at any time, but it does not notify the target session that the debug session is detaching itself, and it does not abort execution of the target session. Therefore, care should be taken to ensure that the target session does not hang itself.

Syntax

```
DBMS_DEBUG.DETACH_SESSION;
```

GET_RUNTIME_INFO Function

This function returns information about the current program. It is only needed if the info_requested parameter to SYNCHRONIZE or CONTINUE was set to 0.

Note: This is currently only used by client-side PL/SQL.

Syntax

```
DBMS_DEBUG.GET_RUNTIME_INFO (
```

```
   info_requested  IN  BINARY_INTEGER,
   run_info        OUT runtime_info)
 RETURN BINARY_INTEGER;
```

Parameters

Table 10–35 GET_RUNTIME_INFO Function Parameters

Parameter	Description
`info_requested`	Which information should be returned in `run_info` when the program stops. See "Information Flags" on page 10-10.
`run_info`	Information about the state of the program.

GET_INDEXES Function

Given a name of a variable or parameter, this function returns the set of its indexes, if it is an indexed table. An error is returned if it is not an indexed table.

Syntax

```
DBMS_DEBUG.GET_INDEXES (
   varname   IN  VARCHAR2,
   frame#    IN  BINARY_INTEGER,
   handle    IN  program_info,
   entries   OUT index_table)
RETURN BINARY_INTEGER;
```

Parameters

Table 10–36 GET_INDEXES Function Parameters

Parameter	Description
`varname`	Name of the variable to get index information about.
`frame#`	Number of frame in which the variable or parameter resides; `NULL` for a package variable.
`handle`	Package description, if object is a package variable.
`entries`	1-based table of the indexes. If non-`NULL`, then `entries`(1) contains the first index of the table, `entries`(2) contains the second index, and so on.

Returns

Table 10–37 GET_INDEXES Function Returns

Return	Description
error_no_such_object	Either: - The package does not exist. - The package is not instantiated. - The user does not have privileges to debug the package. - The object does not exist in the package.

EXECUTE Procedure

This procedure executes SQL or PL/SQL code in the target session. The target session is assumed to be waiting at a breakpoint (or other event). The call to DBMS_DEBUG.EXECUTE occurs in the debug session, which then asks the target session to execute the code.

Syntax

```
DBMS_DEBUG.EXECUTE (
   what         IN VARCHAR2,
   frame#       IN BINARY_INTEGER,
   bind_results IN BINARY_INTEGER,
   results      IN OUT NOCOPY dbms_debug_vc2coll,
   errm         IN OUT NOCOPY VARCHAR2);
```

Parameters

Table 10–38 EXECUTE Procedure Parameters

Parameter	Description
what	SQL or PL/SQL source to execute.
frame#	The context in which to execute the code. Only -1 (global context) is supported at this time.

Table 10–38 EXECUTE Procedure Parameters

Parameter	Description
bind_results	Whether the source wants to bind to results in order to return values from the target session. 0 = No 1 = Yes
results	Collection in which to place results, if bind_results is not 0.
errm	Error message, if an error occurred; otherwise, NULL.

Example 1

This example executes a SQL statement. It returns no results.

```
DECLARE
   coll sys.dbms_debug_vc2coll; -- results (unused)
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute('insert into emp(ename,empno,deptno) ' ||
                      'values(''LJE'', 1, 1)',
                      -1, 0, coll, errm);
END;
```

Example 2

This example executes a PL/SQL block, and it returns no results. The block is an autonomous transaction, which means that the value inserted into the table becomes visible in the debug session.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute(
       'DECLARE PRAGMA autonomous_transaction; ' ||
       'BEGIN ' ||
       '   insert into emp(ename, empno, deptno) ' ||
       '   values(''LJE'', 1, 1); ' ||
       ' COMMIT; ' ||
       'END;',
       -1, 0, coll, errm);
END;
```

Example 3

This example executes a PL/SQL block, and it returns some results.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute(
      'DECLARE ' ||
      '   pp SYS.dbms_debug_vc2coll := SYS.dbms_debug_vc2coll(); ' ||
      '   x  PLS_INTEGER; ' ||
      '   i  PLS_INTEGER := 1; ' ||
      'BEGIN ' ||
      '   SELECT COUNT(*) INTO x FROM emp; ' ||
      '   pp.EXTEND(x * 6); ' ||
      '   FOR c IN (SELECT * FROM emp) LOOP ' ||
      '      pp(i) := ''Ename: '' || c.ename; i := i+1; ' ||
      '      pp(i) := ''Empno: '' || c.empno; i := i+1; ' ||
      '      pp(i) := ''Job:   '' || c.job;   i := i+1; ' ||
      '      pp(i) := ''Mgr:   '' || c.mgr;   i := i+1; ' ||
      '      pp(i) := ''Sal:   '' || c.sal;   i := i+1; ' ||
      '      pp(i) := null;                   i := i+1; ' ||
      '   END LOOP; ' ||
      '   :1 := pp;' ||
      'END;',
       -1, 1, coll, errm);
   each := coll.FIRST;
   WHILE (each IS NOT NULL) LOOP
      dosomething(coll(each));
      each := coll.NEXT(each);
   END LOOP;
END;
```

PRINT_INSTANTIATIONS Procedure

This procedure returns a list of the packages that have been instantiated in the current session.

Syntax

```
DBMS_DEBUG.PRINT_INSTANTIATIONS (
   pkgs  IN OUT NOCOPY backtrace_table,
```

```
   flags  IN BINARY_INTEGER);
```

Parameters

Table 10–39 PRINT_INSTANTIATIONS Procedure Parameters

Parameter	Description
pkgs (OUT)	The instantiated packages
flags	Bitmask of options: ■ 1 - show specs ■ 2 - show bodies ■ 4 - show local instantiations ■ 8 - show remote instantiations (NYI) ■ 16 - do a fast job. The routine does not test whether debug information exists or whether the libunit is shrink-wrapped.

Exceptions

`no_target_program` - target session is not currently executing

Usage Notes

On return, `pkgs` contains a `program_info` for each instantiation. The valid fields are: `Namespace, Name, Owner,` and `LibunitType.`

In addition, Line# contains a bitmask of:

- 1 - the libunit contains debug info
- 2 - the libunit is shrink-wrapped

TARGET_PROGRAM_RUNNING Procedure

This procedure returns TRUE if the target session is currently executing a stored procedure, or FALSE if it is not.

Syntax

FUNCTION target_program_running RETURN BOOLEAN;

PING Procedure

This procedure pings the target session, to prevent it from timing out. Use this procedure when execution is suspended in the target session, for example at a breakpoint.

If the `timeout_behavior` is set to `retry_on_timeout` then this procedure is not necessary.

Syntax

DBMS_DEBUG.PING;

Exceptions

Oracle will display the `no_target_program` exception if there is no target program or if the target session is not currently waiting for input from the debug session.

Timeout Options

Timeout options for the target session are registered with the target session by calling `set_timeout_behavior.`

- `retry_on_timeout` - Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
- `continue_on_timeout` - Continue execution, using same event flags.
- `nodebug_on_timeout` - Turn debug-mode OFF (in other words, call debug_off) and then continue execution. No more events will be generated by this target session unless it is re-initialized by calling debug_on.
- `abort_on_timeout` - Continue execution, using the abort_execution flag, which should cause the program to abort immediately. The session remains in debug-mode.

```
retry_on_timeout CONSTANT BINARY_INTEGER:= 0;

continue_on_timeout CONSTANT BINARY_INTEGER:= 1;

nodebug_on_timeout CONSTANT BINARY_INTEGER:= 2;

abort_on_timeout CONSTANT BINARY_INTEGER:= 3;
```

SET_TIMEOUT_BEHAVIOR Procedure

This procedure tells Probe what to do with the target session when a timeout occurs. This call is made in the target session.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT_BEHAVIOR (
   behavior IN PLS_INTEGER);
```

Parameters

Table 10–40 SET_TIMEOUT_BEHAVIOR Procedure Parameters

Parameter	Description
`behavior` - One of the following:	
▪ `retry_on_timeout`	Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
▪ `continue_on_timeout`	Continue execution, using same event flags.
▪ `nodebug_on_timeout`	Turn debug-mode OFF (in other words, call `debug_off`) and continue execution. No more events will be generated by this target session unless it is re-initialized by calling `debug_on`.
▪ `abort_on_timeout`	Continue execution, using the `abort_execution` flag, which should cause the program to abort immediately. The session remains in debug-mode.

Exceptions

unimplemented - the requested behavior is not recognized

Usage Notes

The default behavior (if this procedure is not called) is `continue_on_timeout`, since it allows a debugger client to reestablish control (at the next event) but does not cause the target session to hang indefinitely.

GET_TIMEOUT_BEHAVIOR Function

This procedure returns the current timeout behavior. This call is made in the target session.

Syntax

```
DBMS_DEBUG.GET_TIMEOUT_BEHAVIOR (
RETURN BINARY_INTEGER;
```

Information Flags

```
info_getOerInfo CONSTANT PLS_INTEGER:= 32;
```

Reasons

```
reason_oer_breakpoint CONSTANT BINARY_INTEGER:= 26;
```

RUNTIME_INFO

`Runtime_info` gives context information about the running program.

Probe v2.4:

Added OER. It gets set if `info_getOerInfo` is set. The OER is a positive number. It can be translated into SQLCODE by translating 1403 to 100, 6510 to 1, and negating any other value.

```
TYPE runtime_info IS RECORD
   (
       Line#            BINARY_INTEGER,   (duplicate of program.line#)
       Terminated       BINARY_INTEGER,   has the program terminated?
       Breakpoint       BINARY_INTEGER,   breakpoint number
       StackDepth       BINARY_INTEGER,   number of frames on the stack
       InterpreterDepth BINARY_INTEGER,   <reserved field>
       Reason           BINARY_INTEGER,   reason for suspension
       Program          program_info,     source location
Following fields were added in Probe v2.4 oer             PLS_INTEGER       OER
(exception), if any
   );
```

oer_table

Used by show_breakpoints

TYPE oer_table IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;

- SET_OER_BREAKPOINT

Set a breakpoint on an OER. The breakpoint persists for the session (or until deleted), as with code breakpoints.

Parameters

Table 10–41

Parameter	Description
oer	The OER (a 4-byte positive number).

Returns

success

Usage Notes

Less functionality is supported on OER breakpoints than on code breakpoints. In particular, note that:

- No "breakpoint number" is returned - the number of the OER is used instead. Thus it is impossible to set duplicate breakpoints on a given OER (it is a no-op).
- It is not possible to disable an OER breakpoint (although clients are free to simulate this by deleting it).
- OER breakpoints are deleted using `delete_oer_breakpoint`.

SET_OER_BREAKPOINT Function

This function sets an OER breakpoint.

Syntax

```
DBMS_DEBUG.SET_OER_BREAKPOINT (
   oer  IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 10–42 SET_OER_BREAKPOINT Function Parameters

Parameter	Description
oer	The OER (positive 4-byte number) to delete.

Returns

success

error_no_such_breakpt - no such OER breakpoint exists

DELETE_OER_BREAKPOINT Function

This function deletes an OER breakpoint.

Syntax

```
DBMS_DEBUG.DELETE_OER_BREAKPOINT (
   oer  IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

SHOW_BREAKPOINTS Procedure

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   code_breakpoints  OUT breakpoint_table,
   oer_breakpoints   OUT oer_table);
```

Parameters

Table 10–43 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
code_breakpoints	The indexed table of breakpoint entries, indexed by breakpoint number.
oer_breakpoints	The indexed table of OER breakpoints, indexed by OER.

- code_breakpoints - indexed table of breakpoint entries, indexed by breakpoint number.
- oer_breakpoints - indexed table of OER breakpoints, indexed by OER.
- PROCEDURE show_breakpoints (code_breakpoints OUT breakpoint_table, oer_breakpoints OUT oer_table);

11

DBMS_DEFER

DBMS_DEFER is the user interface to a replicated transactional deferred remote procedure call facility. Replicated applications use the calls in this interface to queue procedure calls for later transactional execution at remote nodes.

These procedures are typically called from either after row triggers or application specified update procedures.

This chapter discusses the following topics:

- Summary of DBMS_DEFER Subprograms

Summary of DBMS_DEFER Subprograms

Table 11–1 DBMS_DEFER Package Subprograms

Subprogram	Description
CALL Procedure on page 11-2	Builds a deferred call to a remote procedure.
COMMIT_WORK Procedure on page 11-3	Performs a transaction commit after checking for well-formed deferred remote procedure calls.
datatype_ARG Procedure on page 11-4	Provides the data that is to be passed to a deferred remote procedure call.
TRANSACTION Procedure on page 11-6	Indicates the start of a new deferred transaction.

CALL Procedure

This procedure builds a deferred call to a remote procedure.

Syntax

```
DBMS_DEFER.CALL (
   schema_name       IN   VARCHAR2,
   package_name      IN   VARCHAR2,
   proc_name         IN   VARCHAR2,
   arg_count         IN   NATURAL,
   { nodes           IN   node_list_t
   | group_name      IN   VARCHAR2 :=''});
```

Note: This procedure is overloaded. The `nodes` and `group_name` parameters are mutually exclusive.

Parameters

Table 11–2 CALL Procedure Parameters

Parameter	Description
schema_name	Name of the schema in which the stored procedure is located.
package_name	Name of the package containing the stored procedure. The stored procedure must be part of a package. Deferred calls to standalone procedures are not supported.
proc_name	Name of the remote procedure to which you want to defer a call.
arg_count	Number of parameters for the procedure. You must have one call to DBMS_DEFER.*datatype*_ARG for each of these parameters. **Note:** You must include all of the parameters for the procedure, even if some of the parameters have defaults.
nodes	A PL/SQL index-by table of fully qualified database names to which you want to propagate the deferred call. The table is indexed starting at position 1 and continuing until a NULL entry is found, or the no_data_found exception is raised. The data in the table is case insensitive. This parameter is optional.
group_name	Reserved for internal use.

Exceptions

Table 11–3 CALL Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous call was not correctly formed.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Destination list (specified by nodes or by a previous DBMS_DEFER.TRANSACTION call) contains duplicates.

COMMIT_WORK Procedure

This procedure performs a transaction commit after checking for well-formed deferred remote procedure calls.

Syntax

```
DBMS_DEFER.COMMIT_WORK (
```

```
   commit_work_comment IN VARCHAR2);
```

Parameters

Table 11–4 COMMIT_WORK Procedure Parameters

Parameter	Description
commit_work_comment	Equivalent to the COMMIT COMMENT statement in SQL.

Exceptions

Table 11–5 COMMIT_WORK Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Transaction was not correctly formed or terminated.

*datatype*_ARG Procedure

This procedure provides the data that is to be passed to a deferred remote procedure call. Depending upon the type of the data that you need to pass to a procedure, you must call one of the following procedures for each argument to the procedure.

You must specify each parameter in your procedure using the *datatype*_ARG procedure after you execute DBMS_DEFER.CALL. That is, you cannot use the default parameters for the deferred remote procedure call. For example, suppose you have the following procedure:

```
CREATE OR REPLACE PACKAGE my_pack AS
   PROCEDURE my_proc(a VARCHAR2, b VARCHAR2 DEFAULT 'SALES');
END;
/
```

When you run the DBMS_DEFER.CALL procedure, you must include a separate procedure call for each parameter in the my_proc procedure:

```
CREATE OR REPLACE PROCEDURE load_def_tx IS
   node DBMS_DEFER.NODE_LIST_T;
BEGIN
   node(1) := 'MYCOMPUTER.WORLD';
   node(2) := NULL;
   DBMS_DEFER.TRANSACTION(node);
```

```
   DBMS_DEFER.CALL('PR', 'MY_PACK', 'MY_PROC', 2);
   DBMS_DEFER.VARCHAR2_ARG('TEST');
   DBMS_DEFER.VARCHAR2_ARG('SALES'); -- required, cannot omit to use default
END;
/
```

Note:

- The AnyData_ARG procedure supports the following user-defined types: object types, collections, and REFs. See *Oracle9i SQL Reference* for more information about the AnyData datatype.
- This procedure uses abbreviations for some datetime and interval datatypes. For example, TSTZ is used for the TIMESTAMP WITH TIME ZONE datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 1-6.

Syntax

```
DBMS_DEFER.AnyData_ARG      (arg  IN SYS.AnyData);
DBMS_DEFER.NUMBER_ARG       (arg  IN NUMBER);
DBMS_DEFER.DATE_ARG         (arg  IN DATE);
DBMS_DEFER.VARCHAR2_ARG     (arg  IN VARCHAR2);
DBMS_DEFER.CHAR_ARG         (arg  IN CHAR);
DBMS_DEFER.ROWID_ARG        (arg  IN ROWID);
DBMS_DEFER.RAW_ARG          (arg  IN RAW);
DBMS_DEFER.BLOB_ARG         (arg  IN BLOB);
DBMS_DEFER.CLOB_ARG         (arg  IN CLOB);
DBMS_DEFER.NCLOB_ARG        (arg  IN NCLOB);
DBMS_DEFER.NCHAR_ARG        (arg  IN NCHAR);
DBMS_DEFER.NVARCHAR2_ARG    (arg  IN NVARCHAR2);
DBMS_DEFER.ANY_CLOB_ARG     (arg  IN CLOB);
DBMS_DEFER.ANY_VARCHAR2_ARG (arg  IN VARCHAR2);
DBMS_DEFER.ANY_CHAR_ARG     (arg  IN CHAR);
DBMS_DEFER.IDS_ARG          (arg  IN DSINTERVAL_UNCONSTRAINED);
DBMS_DEFER.IYM_ARG          (arg  IN YMINTERVAL_UNCONSTRAINED);
DBMS_DEFER.TIMESTAMP_ARG    (arg  IN TIMESTAMP_UNCONSTRAINED);
DBMS_DEFER.TSLTZ_ARG        (arg  IN TIMESTAMP_LTZ_UNCONSTRAINED);
DBMS_DEFER.TSTZ_ARG         (arg  IN TIMESTAMP_TZ_UNCONSTRAINED);
```

Parameters

Table 11–6 datatype_ARG Procedure Parameters

Parameter	Description
arg	Value of the parameter that you want to pass to the remote procedure to which you previously deferred a call.

Exceptions

Table 11–7 datatype_ARG Procedure Exceptions

Exception	Description
ORA-23323	Argument value is too long.

TRANSACTION Procedure

This procedure indicates the start of a new deferred transaction. If you omit this call, then Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

Syntax

```
DBMS_DEFER.TRANSACTION (
   nodes  IN   node_list_t);
```

Note: This procedure is overloaded. The behavior of the version without an input parameter is similar to that of the version with an input parameter, except that the former uses the nodes in the DEFDEFAULTDEST view instead of using the nodes in the nodes parameter.

Parameters

Table 11–8 TRANSACTION Procedure Parameters

Parameter	Description
nodes	A PL/SQL index-by table of fully qualified database names to which you want to propagate the deferred calls of the transaction. The table is indexed starting at position 1 and continuing until a NULL entry is found, or the no_data_found exception is raised. The data in the table is case insensitive.

Exceptions

Table 11–9 TRANSACTION Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous transaction was not correctly formed or terminated.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Raised by DBMS_DEFER.CALL if the node list contains duplicates.

12

DBMS_DEFER_QUERY

DBMS_DEFER_QUERY enables you to query the deferred transactions queue data that is not exposed through views.

This chapter discusses the following topics:

- Summary of DBMS_DEFER_QUERY Subprograms

Summary of DBMS_DEFER_QUERY Subprograms

Table 12–1 DBMS_DEFER_QUERY Package Subprograms

Subprogram	Description
GET_ARG_FORM Function on page 12-2	Determines the form of an argument in a deferred call.
GET_ARG_TYPE Function on page 12-3	Determines the type of an argument in a deferred call.
GET_CALL_ARGS Procedure on page 12-6	Returns the text version of the various arguments for the specified call.
GET_datatype_ARG Function on page 12-7	Determines the value of an argument in a deferred call.
GET_OBJECT_NULL_ VECTOR_ARG Function on page 12-9	Returns the type information for a column object.

GET_ARG_FORM Function

This function returns the character set form of a deferred call parameter.

> **See Also:** The Replication Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool

Syntax

```
DBMS_DEFER_QUERY.GET_ARG_FORM (
   callno                IN   NUMBER,
   arg_no                IN   NUMBER,
   deferred_tran_id      IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 12–2 GET_ARG_FORM Function Parameters

Parameter	Description
callno	Call identifier from the DEFCALL view.
arg_no	Position of desired parameter in calls argument list. Parameter positions are 1...*number* of parameters in call.

Table 12–2 GET_ARG_FORM Function Parameters

Parameter	Description
deferred_tran_id	Deferred transaction identification.

Exceptions

Table 12–3 GET_ARG_FORM Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 12–4 GET_ARG_FORM Function Returns

Constant Return Value	Return Value	Possible Datatype
DBMS_DEFER_QUERY.ARG_FORM_NONE	0	DATE
		NUMBER
		ROWID
		RAW
		BLOB
		User-defined types
DBMS_DEFER_QUERY.ARG_FORM_IMPLICIT	1	CHAR
		VARCHAR2
		CLOB
DBMS_DEFER_QUERY.ARG_FORM_NCHAR	2	NCHAR
		NVARCHAR2
		NCLOB

GET_ARG_TYPE Function

This function determines the type of an argument in a deferred call. The type of the deferred remote procedure call (RPC) parameter is returned.

See Also: The Replication Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool

Syntax

```
DBMS_DEFER_QUERY.GET_ARG_TYPE (
   callno            IN   NUMBER,
   arg_no            IN   NUMBER,
   deferred_tran_id  IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 12–5 GET_ARG_TYPE Function Parameters

Parameter	Description
callno	Identification number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose type you want to determine. The first argument to a procedure is in position 1.
deferred_tran_id	Identifier of the deferred transaction.

Exceptions

Table 12–6 GET_ARG_TYPE Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 12–7 GET_ARG_TYPE Function Returns

Constant Return Value	Return Value	Corresponding Datatype
DBMS_DEFER_QUERY.ARG_TYPE_VARCHAR2	1	VARCHAR2
DBMS_DEFER_QUERY.ARG_TYPE_NUM	2	NUMBER
DBMS_DEFER_QUERY.ARG_TYPE_ROWID	11	ROWID
DBMS_DEFER_QUERY.ARG_TYPE_DATE	12	DATE
DBMS_DEFER_QUERY.ARG_TYPE_RAW	23	RAW
DBMS_DEFER_QUERY.ARG_TYPE_CHAR	96	CHAR
DBMS_DEFER_QUERY.ARG_TYPE_AnyData	109	AnyData
DBMS_DEFER_QUERY.ARG_TYPE_CLOB	112	CLOB
DBMS_DEFER_QUERY.ARG_TYPE_BLOB	113	BLOB
DBMS_DEFER_QUERY.ARG_TYPE_BFIL	114	BFILE
DBMS_DEFER_QUERY.ARG_TYPE_OBJECT_NULL_VECTOR	121	OBJECT_NULL_VECTOR
DBMS_DEFER_QUERY.ARG_TYPE_TIMESTAMP	180	TIMESTAMP
DBMS_DEFER_QUERY.ARG_TYPE_TSTZ	181	TSTZ
DBMS_DEFER_QUERY.ARG_TYPE_IYM	182	IYM
DBMS_DEFER_QUERY.ARG_TYPE_IDS	183	IDS
DBMS_DEFER_QUERY.ARG_TYPE_TSLTZ	231	TSLTZ

Note:

- The AnyData datatype supports the following user-defined types: object types, collections, and REFs. See *Oracle9i SQL Reference* for more information about the AnyData datatype.
- This function uses abbreviations for some datetime and interval datatypes. For example, TSTZ is used for the TIMESTAMP WITH TIME ZONE datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 1-6.

GET_CALL_ARGS Procedure

This procedure returns the text version of the various arguments for the specified call. The text version is limited to the first 2000 bytes.

See Also:

- "GET_datatype_ARG Function" on page 12-7
- *Oracle9i SQL Reference* for more information about the AnyData datatype

Syntax

```
DBMS_DEFER_QUERY.GET_CALL_ARGS (
   callno    IN  NUMBER,
   startarg  IN  NUMBER := 1,
   argcnt    IN  NUMBER,
   argsize   IN  NUMBER,
   tran_id   IN  VARCHAR2,
   date_fmt  IN  VARCHAR2,
   types     OUT TYPE_ARY,
   forms     OUT TYPE_ARY,
   vals      OUT VAL_ARY);
```

Parameters

Table 12–8 GET_CALL_ARGS Procedure Parameters

Parameter	Description
callno	Identification number from the DEFCALL view of the deferred remote procedure call (RPC).
startarg	Numerical position of the first argument you want described.
argcnt	Number of arguments in the call.
argsize	Maximum size of returned argument.
tran_id	Identifier of the deferred transaction.
date_fmt	Format in which the date is returned.
types	Array containing the types of arguments.
forms	Array containing the character set forms of arguments.
vals	Array containing the values of the arguments in a textual form.

Exceptions

Table 12–9 GET_CALL_ARGS Procedure Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

GET_*datatype*_ARG Function

This function determines the value of an argument in a deferred call.

The AnyData type supports the following user-defined types: object types, collections and REFs. Not all types supported by this function can be enqueued by the AnyData_ARG procedure in the DBMS_DEFER package.

The returned text for type arguments includes the following values: type owner, type name, type version, length, precision, scale, character set identifier, character set form, and number of elements for collections or number of attributes for object types. These values are separated by a colon (:).

See Also:

- "datatype_ARG Procedure" on page 11-4
- The Replication Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool
- *Oracle9i SQL Reference* for more information about the `AnyData` datatype
- This function uses abbreviations for some datetime and interval datatypes. For example, `TSTZ` is used for the `TIMESTAMP WITH TIME ZONE` datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 1-6.

Syntax

Depending upon the type of the argument value that you want to retrieve, the syntax for the appropriate function is as follows. Each of these functions returns the value of the specified argument.

```
DBMS_DEFER_QUERY.GET_datatype_ARG (
   callno             IN   NUMBER,
   arg_no             IN   NUMBER,
   deferred_tran_id   IN   VARCHAR2 DEFAULT NULL)
  RETURN datatype;
```

where *datatype* is:

```
{ AnyData
| NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| ROWID
| BLOB
| CLOB
| NCLOB
| NCHAR
| NVARCHAR2
| IDS
| IYM
| TIMESTAMP
```

```
| TSLTZ
| TSTZ }
```

Parameters

Table 12–10 GET_datatype_ARG Function Parameters

Parameter	Description
callno	Identification number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose value you want to determine. The first argument to a procedure is in position 1.
deferred_tran_id	Identifier of the deferred transaction. Defaults to the last transaction identifier passed to the GET_ARG_TYPE function. The default is NULL.

Exceptions

Table 12–11 GET_datatype_ARG Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.
ORA-26564	Argument in this position is not of the specified type or is not one of the types supported by the AnyData type.

GET_OBJECT_NULL_VECTOR_ARG Function

This function returns the type information for a column object, including the type owner, name, and hashcode.

Syntax

```
DBMS_DEFER_QUERY.GET_OBJECT-NULL_VECTOR_ARG (
   callno                IN   NUMBER,
   arg_no                IN   NUMBER,
   deferred_tran_id      IN   VARCHAR2)
  RETURN SYSTEM.REPCAT$_OBJECT_NULL_VECTOR;
```

Parameters

Table 12–12 GET_OBJECT_NULL_VECTOR_ARG Function Parameters

Parameter	Description
callno	Call identifier from the DEFCALL view.
arg_no	Position of desired parameter in calls argument list. Parameter positions are 1...*number* of parameters in call.
deferred_tran_id	Deferred transaction identification.

Exceptions

Table 12–13 GET_OBJECT_NULL_VECTOR_ARG Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.
ORA-26564	Parameter is not an object_null_vector type.

Returns

Table 12–14 GET_OBJECT_NULL_VECTOR_ARG Function Returns

Return Value	Type Definition

SYSTEM.REPCAT$_OBJECT_NULL_VECTOR type	CREATE TYPE
	SYSTEM.REPCAT$_OBJECT_NULL_VECTOR
	AS OBJECT (
	  type_owner VARCHAR2(30),
	  type_name VARCHAR2(30),
	  type_hashcode RAW(17),
	  null_vector RAW(2000));

13

DBMS_DEFER_SYS

DBMS_DEFER_SYS procedures manage default replication node lists. This package is the system administrator interface to a replicated transactional deferred remote procedure call facility. Administrators and replication daemons can execute transactions queued for remote nodes using this facility, and administrators can control the nodes to which remote calls are destined.

This chapter discusses the following topics:

Summary of DBMS_DEFER_SYS Subprograms

Table 13–1 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
ADD_DEFAULT_DEST Procedure on page 13-3	Adds a destination database to the DEFDEFAULTDEST view.
CLEAR_PROP_ STATISTICS Procedure on page 13-4	Clears the propagation statistics in the DEFSCHEDULE data dictionary view.
DELETE_DEFAULT_DEST Procedure on page 13-5	Removes a destination database from the DEFDEFAULTDEST view.
DELETE_DEF_ DESTINATION Procedure on page 13-5	Removes a destination database from the DEFSCHEDULE view.
DELETE_ERROR Procedure on page 13-6	Deletes a transaction from the DEFERROR view.
DELETE_TRAN Procedure on page 13-6	Deletes a transaction from the DEFTRANDEST view.
DISABLED Function on page 13-7	Determines whether propagation of the deferred transaction queue from the current site to a specified site is enabled.
EXCLUDE_PUSH Function on page 13-8	Acquires an exclusive lock that prevents deferred transaction PUSH.
EXECUTE_ERROR Procedure on page 13-9	Reexecutes a deferred transaction that did not initially complete successfully in the security context of the original receiver of the transaction.
EXECUTE_ERROR_AS_ USER Procedure on page 13-10	Reexecutes a deferred transaction that did not initially complete successfully in the security context of the user who executes this procedure.
PURGE Function on page 13-11	Purges pushed transactions from the deferred transaction queue at your current master site or materialized view site.
PUSH Function on page 13-13	Forces a deferred remote procedure call queue at your current master site or materialized view site to be pushed to a remote site.
REGISTER_ PROPAGATOR Procedure on page 13-17	Registers the specified user as the propagator for the local database.

Table 13–1 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
SCHEDULE_PURGE Procedure on page 13-17	Schedules a job to purge pushed transactions from the deferred transaction queue at your current master site or materialized view site.
SCHEDULE_PUSH Procedure on page 13-19	Schedules a job to push the deferred transaction queue to a remote site.
SET_DISABLED Procedure on page 13-21	Disables or enables propagation of the deferred transaction queue from the current site to a specified destination site.
UNREGISTER_ PROPAGATOR Procedure on page 13-23	Unregisters a user as the propagator from the local database.
UNSCHEDULE_PURGE Procedure on page 13-24	Stops automatic purges of pushed transactions from the deferred transaction queue at a master site or materialized view site.
UNSCHEDULE_PUSH Procedure on page 13-24	Stops automatic pushes of the deferred transaction queue from a master site or materialized view site to a remote site.

ADD_DEFAULT_DEST Procedure

This procedure adds a destination database to the DEFDEFAULTDEST data dictionary view.

Syntax

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST (
   dblink   IN   VARCHAR2);
```

Parameters

Table 13–2 ADD_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to add to the DEFDEFAULTDEST view.

Exceptions

Table 13–3 ADD_DEFAULT_DEST Procedure Exceptions

Exception	Description
ORA-23352	The dblink that you specified is already in the default list.

CLEAR_PROP_STATISTICS Procedure

This procedure clears the propagation statistics in the DEFSCHEDULE data dictionary view. When this procedure is executed successfully, all statistics in this view are returned to zero and statistic gathering starts fresh.

Specifically, this procedure clears statistics from the following columns in the DEFSCHEDULE data dictionary view:

- TOTAL_TXN_COUNT
- AVG_THROUGHPUT
- AVG_LATENCY
- TOTAL_BYTES_SENT
- TOTAL_BYTES_RECEIVED
- TOTAL_ROUND_TRIPS
- TOTAL_ADMIN_COUNT
- TOTAL_ERROR_COUNT
- TOTAL_SLEEP_TIME

Syntax

```
DBMS_DEFER_SYS.CLEAR_PROP_STATISTICS (
   dblink   IN   VARCHAR2);
```

Parameters

Table 13–4 CLEAR_PROP_STATISTICS Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node whose statistics you want to clear. The statistics to be cleared are the statistics for propagation of deferred transactions from the current node to the node you specify for dblink.

DELETE_DEFAULT_DEST Procedure

This procedure removes a destination database from the DEFDEFAULTDEST view.

Syntax

```
DBMS_DEFER_SYS.DELETE_DEFAULT_DEST (
   dblink   IN   VARCHAR2);
```

Parameters

Table 13–5 DELETE_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to delete from the DEFDEFAULTDEST view. If Oracle does not find this dblink in the view, then no action is taken.

DELETE_DEF_DESTINATION Procedure

This procedure removes a destination database from the DEFSCHEDULE view.

Syntax

```
DBMS_DEFER_SYS.DELETE_DEF_DESTINATION (
   destination   IN   VARCHAR2,
   force         IN   BOOLEAN := false);
```

Parameters

Table 13–6 DELETE_DEF_DESTINATION Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the destination that you want to delete from the DEFSCHEDULE view. If Oracle does not find this destination in the view, then no action is taken.
force	When set to true, Oracle ignores all safety checks and deletes the destination.

DELETE_ERROR Procedure

This procedure deletes a transaction from the DEFERROR view.

Syntax

```
DBMS_DEFER_SYS.DELETE_ERROR(
   deferred_tran_id     IN   VARCHAR2,
   destination          IN   VARCHAR2);
```

Parameters

Table 13–7 DELETE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	Identification number from the DEFERROR view of the deferred transaction that you want to remove from the DEFERROR view. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed from the DEFERROR view.

DELETE_TRAN Procedure

This procedure deletes a transaction from the DEFTRANDEST view. If there are no other DEFTRANDEST or DEFERROR entries for the transaction, then the transaction is deleted from the DEFTRAN and DEFCALL views as well.

Syntax

```
DBMS_DEFER_SYS.DELETE_TRAN (
   deferred_tran_id     IN   VARCHAR2,
   destination          IN   VARCHAR2);
```

Parameters

Table 13–8 DELETE_TRAN Procedure Parameters

Parameter	Description
deferred_tran_id	Identification number from the DEFTRAN view of the deferred transaction that you want to delete. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.
destination	The fully qualified database name from the DEFTRANDEST view of the database to which the transaction was originally queued. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.

DISABLED Function

This function determines whether propagation of the deferred transaction queue from the current site to a specified site is enabled. The DISABLED function returns true if the deferred remote procedure call (RPC) queue is disabled for the specified destination.

Syntax

```
DBMS_DEFER_SYS.DISABLED (
   destination  IN   VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 13–9 DISABLED Function Parameters

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to check.

Returns

Table 13–10 DISABLED Function Return Values

Value	Description
`true`	Propagation to this site from the current site is disabled.
`false`	Propagation to this site from the current site is enabled.

Exceptions

Table 13–11 DISABLED Function Exceptions

Exception	Description
`NO_DATA_FOUND`	Specified `destination` does not appear in the `DEFSCHEDULE` view.

EXCLUDE_PUSH Function

This function acquires an exclusive lock that prevents deferred transaction PUSH (either serial or parallel). This function performs a commit when acquiring the lock. The lock is acquired with `RELEASE_ON_COMMIT => true`, so that pushing of the deferred transaction queue can resume after the next commit.

Syntax

```
DBMS_DEFER_SYS.EXCLUDE_PUSH (
   timeout   IN   INTEGER)
  RETURN INTEGER;
```

Parameters

Table 13–12 EXCLUDE_PUSH Function Parameters

Parameter	Description
`timeout`	Timeout in seconds. If the lock cannot be acquired within this time period (either because of an error or because a PUSH is currently under way), then the call returns a value of 1. A timeout value of `DBMS_LOCK.MAXWAIT` waits indefinitely.

Returns

Table 13–13 EXCLUDE_PUSH Function Return Values

Value	Description
0	Success, lock acquired.
1	Timeout, no lock acquired.
2	Deadlock, no lock acquired.
4	Already own lock.

EXECUTE_ERROR Procedure

This procedure reexecutes a deferred transaction that did not initially complete successfully in the security context of the original receiver of the transaction.

Syntax

```
DBMS_DEFER_SYS.EXECUTE_ERROR (
   deferred_tran_id IN   VARCHAR2,
   destination      IN   VARCHAR2);
```

Parameters

Table 13–14 EXECUTE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	Identification number from the DEFERROR view of the deferred transaction that you want to reexecute. If this is NULL, then all transactions queued for destination are reexecuted.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL. If the provided database name is not fully qualified or is invalid, no error will be raised.

Exceptions

Table 13–15 EXECUTE_ERROR Procedure Exceptions

Exception	Description
ORA-24275 error	Illegal combinations of NULL and non-NULL parameters were used.

Table 13–15 EXECUTE_ERROR Procedure Exceptions

Exception	Description
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

EXECUTE_ERROR_AS_USER Procedure

This procedure reexecutes a deferred transaction that did not initially complete successfully. Each transaction is executed in the security context of the connected user.

Syntax

```
DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
   deferred_tran_id IN   VARCHAR2,
   destination      IN   VARCHAR2);
```

Parameters

Table 13–16 EXECUTE_ERROR_AS_USER Procedure Parameters

Parameter	Description
deferred_tran_id	Identification number from the DEFERROR view of the deferred transaction that you want to reexecute. If this is NULL, then all transactions queued for destination are reexecuted.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.

Exceptions

Table 13–17 EXECUTE_ERROR_AS_USER Procedure Exceptions

Exception	Description
ORA-24275 error	Illegal combinations of NULL and non-NULL parameters were used.
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

PURGE Function

This function purges pushed transactions from the deferred transaction queue at your current master site or materialized view site.

Syntax

```
DBMS_DEFER_SYS.PURGE (
   purge_method          IN  BINARY_INTEGER := purge_method_quick,
   rollback_segment      IN  VARCHAR2       := NULL,
   startup_seconds       IN  BINARY_INTEGER := 0,
   execution_seconds     IN  BINARY_INTEGER := seconds_infinity,
   delay_seconds         IN  BINARY_INTEGER := 0,
   transaction_count     IN  BINARY_INTEGER := transactions_infinity,
   write_trace           IN  BOOLEAN        := NULL);
  RETURN BINARY_INTEGER;
```

Parameters

Table 13–18 PURGE Function Parameters

Parameter	Description
purge_method	Controls how to purge the deferred transaction queue: purge_method_quick costs less, while purge_method_precise offers better precision. Specify the following for this parameter to use purge_method_quick: dbms_defer_sys.purge_method_quick Specify the following for this parameter to user purge_method_precise: dbms_defer_sys.purge_method_precise If you use purge_method_quick, deferred transactions and deferred procedure calls that have been successfully pushed may remain in the DEFTRAN and DEFCALL data dictionary views for longer than expected before they are purged. See "Usage Notes" on page 13-13 for more information.
rollback_segment	Name of rollback segment to use for the purge, or NULL for default.
startup_seconds	Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.
execution_seconds	If > 0, then stop purge cleanly after the specified number of seconds of real time.

Table 13–18 PURGE Function Parameters

Parameter	Description
delay_seconds	Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.
transaction_count	If > 0, then shut down cleanly after purging transaction_count number of transactions.
write_trace	When set to true, Oracle records the result value returned by the PURGE function in the server's trace file. When set to false, Oracle does not record the result value.

Returns

Table 13–19 Purge Function Returns

Value	Description
result_ok	OK, terminated after delay_seconds expired.
result_startup_seconds	Terminated by lock timeout while starting.
result_execution_seconds	Terminated by exceeding execution_seconds.
result_transaction_count	Terminated by exceeding transaction_count.
result_errors	Terminated after errors.
result_split_del_order_limit	Terminated after failing to acquire the enqueue in exclusive mode. If you receive this return code, then retry the purge. If the problem persists, then contact Oracle Support Services.
result_purge_disabled	Queue purging is disabled internally for synchronization when adding new master sites without quiesce.

Exceptions

Table 13–20 PURGE Function Exceptions

Exception	Description
argoutofrange	Parameter value is out of a valid range.
executiondisabled	Execution of purging is disabled.
defererror	Internal error.

Usage Notes

When you use the purge_method_quick for the purge_method parameter in the DBMS_DEFER_SYS.PURGE function, deferred transactions and deferred procedure calls may remain in the DEFCALL and DEFTRAN data dictionary views after they have been successfully pushed. This behavior occurs in replication environments that have more than one database link and the push is executed to only one database link.

To purge the deferred transactions and deferred procedure calls, perform one of the following actions:

- Use purge_method_precise for the purge_method parameter instead of the purge_method_quick. Using purge_method_precise is more expensive, but it ensures that the deferred transactions and procedure calls are purged after they have been successfully pushed.
- Using purge_method_quick for the purge_method parameter, push the deferred transactions to all database links. The deferred transactions and deferred procedure calls are purged efficiently when the push to the last database link is successful.

PUSH Function

This function forces a deferred remote procedure call (RPC) queue at your current master site or materialized view site to be pushed (propagated) to a remote site using either serial or parallel propagation.

Syntax

```
DBMS_DEFER_SYS.PUSH (
   destination          IN  VARCHAR2,
   parallelism          IN  BINARY_INTEGER := 0,
   heap_size            IN  BINARY_INTEGER := 0,
   stop_on_error        IN  BOOLEAN        := false,
```

```
  write_trace          IN  BOOLEAN        := false,
  startup_seconds      IN  BINARY_INTEGER := 0,
  execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
  delay_seconds        IN  BINARY_INTEGER := 0,
  transaction_count    IN  BINARY_INTEGER := transactions_infinity,
  delivery_order_limit IN  NUMBER         := delivery_order_infinity)
RETURN BINARY_INTEGER;
```

Parameters

Table 13–21 PUSH Function Parameters

Parameter	Description
`destination`	The fully qualified database name of the master site or master materialized view site to which you are forwarding changes.
`parallelism`	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
`heap_size`	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set the parameter unless so directed by Oracle Support Services.
`stop_on_error`	The default, `false`, indicates that the executor should continue even if errors, such as conflicts, are encountered. If `true`, then stops propagation at the first indication that a transaction encountered an error at the destination site.
`write_trace`	When set to `true`, Oracle records the result value returned by the function in the server's trace file. When set to `false`, Oracle does not record the result value.
`startup_seconds`	Maximum number of seconds to wait for a previous push to the same destination.

Table 13–21 PUSH Function Parameters

Parameter	Description
execution_seconds	If > 0, then stop push cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue. The execution_seconds parameter only controls the duration of time that operations can be started. It does not include the amount of time that the transactions require at remote sites. Therefore, the execution_seconds parameter is not intended to be used as a precise control to stop the propagation of transactions to a remote site. If a precise control is required, use the transaction_count or delivery_order parameters.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.
delivery_order_limit	Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit

Returns

Table 13–22 PUSH Function Returns

Value	Description
result_ok	OK, terminated after delay_seconds expired.
result_startup_seconds	Terminated by lock timeout while starting.
result_execution_seconds	Terminated by exceeding execution_seconds.
result_transaction_count	Terminated by exceeding transaction_count.
result_delivery_order_limit	Terminated by exceeding delivery_order_limit.
result_errors	Terminated after errors.
result_push_disabled	Push was disabled internally. Typically, this return value means that propagation to the destination was set to disabled internally by Oracle for propagation synchronization when adding a new master site to a master group without quiescing the master group. Oracle will enable propagation automatically at a later time
result_split_del_order_limit	Terminated after failing to acquire the enqueue in exclusive mode. If you receive this return code, then retry the push. If the problem persists, then contact Oracle Support Services.

Exceptions

Table 13–23 PUSH Function Exceptions

Exception	Description
incompleteparallelpush	Serial propagation requires that parallel propagation shuts down cleanly.
executiondisabled	Execution of deferred remote procedure calls (RPCs) is disabled at the destination.
crt_err_err	Error while creating entry in DEFERROR.
deferred_rpc_quiesce	Replication activity for replication group is suspended.
commfailure	Communication failure during deferred remote procedure call (RPC).
missingpropagator	A propagator does not exist.

REGISTER_PROPAGATOR Procedure

This procedure registers the specified user as the propagator for the local database. It also grants the following privileges to the specified user (so that the user can create wrappers):

- CREATE SESSION
- CREATE PROCEDURE
- CREATE DATABASE LINK
- EXECUTE ANY PROCEDURE

Syntax

```
DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
   username  IN  VARCHAR2);
```

Parameters

Table 13–24 REGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the user.

Exceptions

Table 13–25 REGISTER_PROPAGATOR Procedure Exceptions

Exception	Description
missinguser	Specified user does not exist.
alreadypropagator	Specified user is already the propagator.
duplicatepropagator	There is already a different propagator.

SCHEDULE_PURGE Procedure

This procedure schedules a job to purge pushed transactions from the deferred transaction queue at your current master site or materialized view site. You should schedule one purge job.

See Also: *Oracle9i Replication* for information about using this procedure to schedule continuous or periodic purge of your deferred transaction queue

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PURGE (
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := NULL,
   purge_method         IN  BINARY_INTEGER := NULL,
   rollback_segment     IN  VARCHAR2       := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL,
   write_trace          IN  BOOLEAN        := NULL);
```

Parameters

Table 13–26 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
`interval`	Allows you to provide a function to calculate the next time to purge. This value is stored in the `interval` field of the `DEFSCHEDULE` view and calculates the `next_date` field of this view. If you use the default value for this parameter, `NULL`, then the value of this field remains unchanged. If the field had no previous value, it is created with a value of `NULL`. If you do not supply a value for this field, you must supply a value for `next_date`.
`next_date`	Allows you to specify a time to purge pushed transactions from the site's queue. This value is stored in the `next_date` field of the `DEFSCHEDULE` view. If you use the default value for this parameter, `NULL`, then the value of this field remains unchanged. If this field had no previous value, it is created with a value of `NULL`. If you do not supply a value for this field, then you must supply a value for `interval`.
`reset`	Set to `true` to reset `LAST_TXN_COUNT`, `LAST_ERROR`, and `LAST_MSG` to `NULL`.

Table 13–26 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
purge_method	Controls how to purge the deferred transaction queue: purge_method_quick costs less, while purge_method_precise offers better precision. Specify the following for this parameter to use purge_method_quick: dbms_defer_sys.purge_method_quick Specify the following for this parameter to user purge_method_precise: dbms_defer_sys.purge_method_precise If you use purge_method_quick, deferred transactions and deferred procedure calls that have been successfully pushed may remain in the DEFTRAN and DEFCALL data dictionary views for longer than expected before they are purged. For more information, see "Usage Notes" on page 13-13. These usage notes are for the DBMS_DEFER_SYS.PURGE function, but they also apply to the DBMS_DEFER_SYS.SCHEDULE_PURGE procedure.
rollback_segment	Name of rollback segment to use for the purge, or NULL for default.
startup_seconds	Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.
execution_seconds	If >0, then stop purge cleanly after the specified number of seconds of real time.
delay_seconds	Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.
transaction_count	If > 0, then shut down cleanly after purging transaction_count number of transactions.
write_trace	When set to true, Oracle records the result value returned by the PURGE function in the server's trace file.

SCHEDULE_PUSH Procedure

This procedure schedules a job to push the deferred transaction queue to a remote site. This procedure performs a COMMIT.

See Also: *Oracle9i Replication* for information about using this procedure to schedule continuous or periodic push of your deferred transaction queue

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PUSH (
   destination          IN  VARCHAR2,
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := false,
   parallelism          IN  BINARY_INTEGER := NULL,
   heap_size            IN  BINARY_INTEGER := NULL,
   stop_on_error        IN  BOOLEAN        := NULL,
   write_trace          IN  BOOLEAN        := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL);
```

Parameters

Table 13–27 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
`destination`	The fully qualified database name of the master site or master materialized view site to which you are forwarding changes.
`interval`	Allows you to provide a function to calculate the next time to push. This value is stored in the `interval` field of the `DEFSCHEDULE` view and calculates the `next_date` field of this view. If you use the default value for this parameter, `NULL`, then the value of this field remains unchanged. If the field had no previous value, it is created with a value of `NULL`. If you do not supply a value for this field, then you must supply a value for `next_date`.
`next_date`	Allows you to specify a time to push deferred transactions to the remote site. This value is stored in the `next_date` field of the `DEFSCHEDULE` view. If you use the default value for this parameter, `NULL`, then the value of this field remains unchanged. If this field had no previous value, then it is created with a value of `NULL`. If you do not supply a value for this field, then you must supply a value for `interval`.

Table 13–27 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
reset	Set to true to reset LAST_TXN_COUNT, LST_ERROR, and LAST_MSG to NULL.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set the parameter unless so directed by Oracle Support Services.
stop_on_error	The default, false, indicates that the executor should continue even if errors, such as conflicts, are encountered. If true, then stops propagation at the first indication that a transaction encountered an error at the destination site.
write_trace	When set to true, Oracle records the result value returned by the function in the server's trace file.
startup_seconds	Maximum number of seconds to wait for a previous push to the same destination.
execution_seconds	If >0, then stop execution cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.

SET_DISABLED Procedure

This procedure disables or enables propagation of the deferred transaction queue from the current site to a specified destination site. If the disabled parameter is true, then the procedure disables propagation to the specified destination and future invocations of PUSH do not push the deferred remote procedure call (RPC) queue. SET_DISABLED eventually affects a session already pushing the queue to

the specified destination, but does not affect sessions appending to the queue with DBMS_DEFER.

If the disabled parameter is false, then the procedure enables propagation to the specified destination and, although this does not push the queue, it permits future invocations of PUSH to push the queue to the specified destination. Whether the disabled parameter is true or false, a COMMIT is required for the setting to take effect in other sessions.

Syntax

```
DBMS_DEFER_SYS.SET_DISABLED (
   destination   IN   VARCHAR2,
   disabled      IN   BOOLEAN := true,
   catchup       IN   RAW := '00',
   override      IN   BOOLEAN := false);
```

Parameters

Table 13–28 SET_DISABLED Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to change.
disabled	By default, this parameter disables propagation of the deferred transaction queue from your current site to the specified destination. Set this to false to enable propagation.
catchup	The extension identifier for adding new master sites to a master group without quiescing the master group. The new master site is the destination. Query the DEFSCHEDULE data dictionary view for the existing extension identifiers.
override	A false setting, the default, specifies that Oracle raises the cantsetdisabled exception if the disabled parameter is set to false and propagation was disabled internally by Oracle. A true setting specifies that Oracle ignores whether the disabled state was set internally for synchronization and always tries to set the state as specified by the disabled parameter. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 13–29 SET_DISABLED Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the specified destination.
cantsetdisabled	The disabled status for this site is set internally by Oracle for synchronization during adding a new master site to a master group without quiescing the master group. Ensure that adding a new master site without quiescing finished before invoking this procedure.

UNREGISTER_PROPAGATOR Procedure

To unregister a user as the propagator from the local database. This procedure:

- Deletes the specified propagator from DEFPROPAGATOR.
- Revokes privileges granted by REGISTER_PROPAGATOR from the specified user (including identical privileges granted independently).
- Drops any generated wrappers in the schema of the specified propagator, and marks them as dropped in the replication catalog.

Syntax

```
DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR (
   username  IN  VARCHAR2
   timeout   IN  INTEGER DEFAULT DBMS_LOCK.MAXWAIT);
```

Parameters

Table 13–30 UNREGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the propagator user.
timeout	Timeout in seconds. If the propagator is in use, then the procedure waits until timeout. The default is DBMS_LOCK.MAXWAIT.

Exceptions

Table 13–31 UNREGISTER_PROPAGATOR Procedure Exceptions

Parameter	Description
missingpropagator	Specified user is not a propagator.
propagator_inuse	Propagator is in use, and thus cannot be unregistered. Try later.

UNSCHEDULE_PURGE Procedure

This procedure stops automatic purges of pushed transactions from the deferred transaction queue at a master site or materialized view site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PURGE();
```

UNSCHEDULE_PUSH Procedure

This procedure stops automatic pushes of the deferred transaction queue from a master site or materialized view site to a remote site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PUSH (
   dblink   IN   VARCHAR2);
```

Parameters

Table 13–32 UNSCHEDULE_PUSH Procedure Parameters

Parameter	Description
dblink	Fully qualified path name for the database at which you want to unschedule periodic execution of deferred remote procedure calls.

Table 13–33 UNSCHEDULE_PUSH Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the specified dblink.

14

DBMS_DESCRIBE

You can use the DBMS_DESCRIBE package to get information about a PL/SQL object. When you specify an object name, DBMS_DESCRIBE returns a set of indexed tables with the results. Full name translation is performed and security checking is also checked on the final object.

This package provides the same functionality as the Oracle Call Interface OCIDescribeAny call.

See Also: *Oracle Call Interface Programmer's Guide*

This chapter discusses the following topics:

- Security, Types, and Errors for DBMS_DESCRIBE
- Summary of DBMS_DESCRIBE Subprograms

Security, Types, and Errors for DBMS_DESCRIBE

Security

This package is available to PUBLIC and performs its own security checking based on the schema object being described.

Types

The DBMS_DESCRIBE package declares two PL/SQL table types, which are used to hold data returned by DESCRIBE_PROCEDURE in its OUT parameters. The types are:

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(30)
    INDEX BY BINARY_INTEGER;

TYPE NUMBER_TABLE IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
```

Errors

DBMS_DESCRIBE can raise application errors in the range -20000 to -20004.

Table 14–1 DBMS_DESCRIBE Errors

Error	Description
ORA-20000	ORU 10035: cannot describe a package ('X') only a procedure within a package.
ORA-20001	ORU-10032: procedure 'X' within package 'Y' does not exist.
ORA-20002	ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.
ORA-20003	ORU-10036: object 'X' is invalid and cannot be described.
ORA-20004	Syntax error attempting to parse 'X'.

Summary of DBMS_DESCRIBE Subprograms

DBMS_DESCRIBE contains only one procedure: DESCRIBE_PROCEDURE.

DESCRIBE_PROCEDURE Procedure

The procedure DESCRIBE_PROCEDURE accepts the name of a stored procedure, a description of the procedure, and each of its parameters.

Syntax

```
DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
   object_name    IN  VARCHAR2,
   reserved1      IN  VARCHAR2,
   reserved2      IN  VARCHAR2,
   overload       OUT NUMBER_TABLE,
   position       OUT NUMBER_TABLE,
   level          OUT NUMBER_TABLE,
   argument_name  OUT VARCHAR2_TABLE,
   datatype       OUT NUMBER_TABLE,
   default_value  OUT NUMBER_TABLE,
   in_out         OUT NUMBER_TABLE,
   length         OUT NUMBER_TABLE,
   precision      OUT NUMBER_TABLE,
   scale          OUT NUMBER_TABLE,
   radix          OUT NUMBER_TABLE,
   spare          OUT NUMBER_TABLE);
```

Parameters

Table 14–2 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
`object_name`	Name of the procedure being described. The syntax for this parameter follows the rules used for identifiers in SQL. The name can be a synonym. This parameter is required and may not be null. The total length of the name cannot exceed 197 bytes. An incorrectly specified `OBJECT_NAME` can result in one of the following exceptions: `ORA-20000` - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function. `ORA-20001` - The procedure or function that you specified does not exist within the given package. `ORA-20002` - The object that you specified is a remote object. This procedure cannot currently describe remote objects. `ORA-20003` - The object that you specified is invalid and cannot be described. `ORA-20004` - The object was specified with a syntax error.
`reserved1` `reserved2`	Reserved for future use -- must be set to `NULL` or the empty string.

Table 14–2 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
overload	A unique number assigned to the procedure's signature. If a procedure is overloaded, then this field holds a different value for each version of the procedure.
position	Position of the argument in the parameter list. Position 0 returns the values for the return type of a function.
level	If the argument is a composite type, such as record, then this parameter returns the level of the datatype. See the *Oracle Call Interface Programmer's Guide* for a description of the ODESSP call for an example.
argument_name	Name of the argument associated with the procedure that you are describing.
datatype	Oracle datatype of the argument being described. The datatypes and their numeric type codes are: 0 placeholder for procedures with no arguments 1 VARCHAR, VARCHAR, STRING 2 NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL 3 BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL 8 LONG 11 ROWID 12 DATE 23 RAW 24 LONG RAW 96 CHAR (ANSI FIXED CHAR), CHARACTER 106 MLSLABEL 250 PL/SQL RECORD 251 PL/SQL TABLE 252 PL/SQL BOOLEAN
default_value	1 if the argument being described has a default value; otherwise, the value is 0.
in_out	Describes the mode of the parameter: 0 IN 1 OUT 2 IN OUT

Table 14–2 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
length	Data length, in bytes, of the argument being described.
precision	If the argument being described is of datatype 2 (NUMBER), then this parameter is the precision of that number.
scale	If the argument being described is of datatype 2 (NUMBER), then this parameter is the scale of that number.
radix	If the argument being described is of datatype 2 (NUMBER), then this parameter is the radix of that number.
spare	Reserved for future functionality.

Return Values

All values from DESCRIBE_PROCEDURE are returned in its OUT parameters. The datatypes for these are PL/SQL tables, in order to accommodate a variable number of parameters.

Using DBMS_DESCRIBE: Examples

One use of the DESCRIBE_PROCEDURE procedure is as an external service interface.

For example, consider a client that provides an OBJECT_NAME of SCOTT.ACCOUNT_UPDATE, where ACCOUNT_UPDATE is an overloaded function with specification:

```
table account (account_no number, person_id number,
               balance number(7,2))
table person  (person_id number(4), person_nm varchar2(10))

function ACCOUNT_UPDATE (account_no   number,
                         person       person%rowtype,
                         amounts      dbms_describe.number_table,
                         trans_date   date)
                         return       accounts.balance%type;

function ACCOUNT_UPDATE (account_no   number,
                         person       person%rowtype,
                         amounts      dbms_describe.number_table,
                         trans_no     number)
                         return       accounts.balance%type;
```

This procedure might look similar to the following output:

```
overload position  argument level  datatype length prec scale rad
-------- --------- -------- ------ -------- ------ ---- ----- ---
       1         0                0        2     22    7     2  10
       1         1   ACCOUNT      0        2      0    0     0   0
       1         2   PERSON       0      250      0    0     0   0
       1         1   PERSON_ID    1        2     22    4     0  10
       1         2   PERSON_NM    1        1     10    0     0   0
       1         3   AMOUNTS      0      251      0    0     0   0
       1         1                1        2     22    0     0   0
       1         4   TRANS_DATE   0       12      0    0     0   0
       2         0                0        2     22    7     2  10
       2         1   ACCOUNT_NO   0        2     22    0     0   0
       2         2   PERSON       0        2     22    4     0  10
       2         3   AMOUNTS      0      251     22    4     0  10
       2         1                1        2      0    0     0   0
       2         4   TRANS_NO     0        2      0    0     0   0
```

The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```
CREATE OR REPLACE PROCEDURE p1 (
        pvc2    IN     VARCHAR2,
        pvc     OUT    VARCHAR,
        pstr    IN OUT STRING,
        plong   IN     LONG,
        prowid  IN     ROWID,
        pchara  IN     CHARACTER,
        pchar   IN     CHAR,
        praw    IN     RAW,
        plraw   IN     LONG RAW,
        pbinint IN     BINARY_INTEGER,
        pplsint IN     PLS_INTEGER,
        pbool   IN     BOOLEAN,
        pnat    IN     NATURAL,
        ppos    IN     POSITIVE,
        pposn   IN     POSITIVEN,
        pnatn   IN     NATURALN,
        pnum    IN     NUMBER,
        pintgr  IN     INTEGER,
        pint    IN     INT,
        psmall  IN     SMALLINT,
        pdec    IN     DECIMAL,
        preal   IN     REAL,
        pfloat  IN     FLOAT,
        pnumer  IN     NUMERIC,
```

```
        pdp     IN     DOUBLE PRECISION,
        pdate   IN     DATE,
        pmls    IN     MLSLABEL) AS

BEGIN
    NULL;
END;
```

If you describe this procedure using the following:

```
CREATE OR REPLACE PACKAGE describe_it AS

    PROCEDURE desc_proc (name VARCHAR2);

END describe_it;

CREATE OR REPLACE PACKAGE BODY describe_it AS

  PROCEDURE prt_value(val VARCHAR2, isize INTEGER) IS
    n INTEGER;
  BEGIN
    n := isize - LENGTHB(val);
    IF n < 0 THEN
      n := 0;
    END IF;
    DBMS_OUTPUT.PUT(val);
    FOR i in 1..n LOOP
      DBMS_OUTPUT.PUT(' ');
    END LOOP;
  END prt_value;

  PROCEDURE desc_proc (name VARCHAR2) IS

      overload     DBMS_DESCRIBE.NUMBER_TABLE;
      position     DBMS_DESCRIBE.NUMBER_TABLE;
      c_level      DBMS_DESCRIBE.NUMBER_TABLE;
      arg_name     DBMS_DESCRIBE.VARCHAR2_TABLE;
      dty          DBMS_DESCRIBE.NUMBER_TABLE;
      def_val      DBMS_DESCRIBE.NUMBER_TABLE;
      p_mode       DBMS_DESCRIBE.NUMBER_TABLE;
      length       DBMS_DESCRIBE.NUMBER_TABLE;
      precision    DBMS_DESCRIBE.NUMBER_TABLE;
      scale        DBMS_DESCRIBE.NUMBER_TABLE;
      radix        DBMS_DESCRIBE.NUMBER_TABLE;
      spare        DBMS_DESCRIBE.NUMBER_TABLE;
```

```
    idx          INTEGER := 0;

BEGIN
    DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
            name,
            null,
            null,
            overload,
            position,
            c_level,
            arg_name,
            dty,
            def_val,
            p_mode,
            length,
            precision,
            scale,
            radix,
            spare);

    DBMS_OUTPUT.PUT_LINE('Position    Name        DTY  Mode');
    LOOP
        idx := idx + 1;
        prt_value(TO_CHAR(position(idx)), 12);
        prt_value(arg_name(idx), 12);
        prt_value(TO_CHAR(dty(idx)), 5);
        prt_value(TO_CHAR(p_mode(idx)), 5);
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.NEW_LINE;

END desc_proc;
END describe_it;
```

Then the results list all the numeric codes for the PL/SQL datatypes:

```
Position  Name    Datatype_Code  Mode
1         PVC2      1              0
2         PVC       1              1
3         PSTR      1              2
4         PLONG     8              0
5         PROWID    11             0
```

```
6          PCHARA     96          0
7          PCHAR      96          0
8          PRAW       23          0
9          PLRAW      24          0
10         PBININT    3           0
11         PPLSINT    3           0
12         PBOOL      252         0
13         PNAT       3           0
14         PPOS       3           0
15         PPOSN      3           0
16         PNATN      3           0
17         PNUM       2           0
18         PINTGR     2           0
19         PINT       2           0
20         PSMALL     2           0
21         PDEC       2           0
22         PREAL      2           0
23         PFLOAT     2           0
24         PNUMER     2           0
25         PDP        2           0
26         PDATE      12          0
27         PMLS       106         0
```

Usage Notes

There is currently no way from a third generation language to directly bind to an argument of type `record` or `boolean`. For Booleans, there are the following work-arounds:

- Assume function `F` returns a Boolean. `G` is a procedure with one `IN` Boolean argument, and `H` is a procedure which has one `OUT` Boolean argument. Then, you can execute these functions, binding in `DTYINTs` (native integer) as follows, where `0=>FALSE` and `1=>TRUE`:

```
begin :dtyint_bind_var := to_number(f); end;

begin g(to_boolean(:dtyint_bind_var)); end;

declare b boolean; begin h(b); if b then :dtyint_bind_var := 1;
else :dtyint_bind_var := 0; end if; end;
```

- Access to procedures with arguments of type `record` require writting a wrapper similar to that in the preceding example (see funciton `H`).

15

DBMS_DISTRIBUTED_TRUST_ADMIN

DBMS_DISTRIBUTED_TRUST_ADMIN procedures maintain the Trusted Servers List. Use these procedures to define whether a server is trusted. If a database is not trusted, Oracle refuses current user database links from the database.

Oracle uses local Trusted Servers Lists, along with enterprise domain membership lists stored in the enterprise LDAP directory service, to determine if another database is trusted. The LDAP directory service entries are managed with the Enterprise Security Manager Tool in Oracle Enterprise Manager.

Oracle considers another database to be "trusted" if it meets the following criteria:

1. It is in the same enterprise domain in the directory service as the local database.
2. The enterprise domain is marked as trusted in the directory service.
3. It is not listed as untrusted in the local Trusted Servers List. Current user database links will only be accepted from another database if both databases involved trust each other.

You can list a database server locally in the Trusted Servers List regardless of what is listed in the directory service. However, if you list a database that is not in the same domain as the local database, or if that domain is untrusted, the entry will have no effect.

This functionality is part of the Enterprise User Security feature of the Oracle Advanced Security Option.

This chapter discusses the following topics:

Requirements

To execute DBMS_DISTRIBUTED_TRUST_ADMIN, the EXECUTE_CATALOG_ROLE role must be granted to the DBA. To select from the view TRUSTED_SERVERS, the SELECT_CATALOG_ROLE role must be granted to the DBA.

It is important to know whether all servers are trusted or not trusted. Trusting a particular server with the ALLOW_SERVER procedure does not have any effect if the database already trusts all databases, or if that database is already trusted. Similarly, denying a particular server with the DENY_SERVER procedure does not have any effect if the database already does not trust any database or if that database is already untrusted.

The procedures DENY_ALL and ALLOW_ALL delete all entries (in other words, server names) that are explicitly allowed or denied using the ALLOW_SERVER procedure or DENY_SERVER procedure respectively.

Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms

Table 15–1 DBMS_DISTRIBUTED_TRUST_ADMIN Package Subprograms

Subprogram	Description
ALLOW_ALL Procedure on page 15-2	Empties the list and inserts a row indicating that all servers should be trusted.
ALLOW_SERVER Procedure on page 15-3	Enables a specific server to be allowed access even though deny all is indicated in the list.
DENY_ALL Procedure on page 15-3	Empties the list and inserts a row indicating that all servers should be untrusted.
DENY_SERVER Procedure on page 15-4	Enables a specific server to be denied access even though allow all is indicated in the list.

ALLOW_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers that are members of a trusted domain in an enterprise directory service and that are in the same domain are allowed access.

The view TRUSTED_SERVERS will show "TRUSTED ALL" indicating that the database trusts all servers that are currently trusted by the enterprise directory service.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL;
```

Usage Notes

ALLOW_ALL only applies to servers listed as trusted in the enterprise directory service and in the same enterprise domain.

ALLOW_SERVER Procedure

This procedure ensures that the specified server is considered trusted (even if you have previously specified "deny all").

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 15–2 ALLOW_SERVER Procedure Parameters

Parameter	Description
server	Unique, fully-qualified name of the server to be trusted.

Usage Notes

If the Trusted Servers List contains the entry "deny all", then this procedure adds a specification indicating that a specific database (for example, DBx) is to be trusted.

If the Trusted Servers List contains the entry "allow all", and if there is no "deny DBx" entry in the list, then executing this procedure causes no change.

If the Trusted Servers List contains the entry "allow all", and if there is a "deny DBx" entry in the list, then that entry is deleted.

DENY_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers are denied access.

The view TRUSTED_SERVERS will show "UNTRUSTED ALL" indicating that no servers are currently trusted.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

DENY_SERVER Procedure

This procedure ensures that the specified server is considered untrusted (even if you have previously specified `allow all`).

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 15–3 DENY_SERVER Procedure Parameters

Parameter	Description
`server`	Unique, fully-qualified name of the server to be untrusted.

Usage Notes

If the Trusted Servers List contains the entry `allow all`, then this procedure adds an entry indicating that the specified database (for example, `DBx`) is not to be trusted.

If the Trusted Servers List contains the entry "`deny all`", and if there is no "`allow DBx`" entry in the list, then this procedure causes no change.

If the Trusted Servers List contains the entry "`deny all`", and if there is an "`allow DBx`" entry, then this procedure causes that entry to be deleted.

Example

If you have not yet used the package `DBMS_DISTRIBUTED_TRUST_ADMIN` to change the trust listing, by default you trust all databases in the same enterprise domain if that domain it listed as trusted in the directory service:

```
SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
---------- --------------------
Trusted    All

1 row selected.
```

Because all servers are currently trusted, you can execute the DENY_SERVER procedure and specify that a particular server is not trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER
                          ('SALES.US.AMERICAS.ACME_AUTO.COM');

Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
--------- ----------------------------------------------
Untrusted SALES.US.AMERICAS.ACME_AUTO.COM

1 row selected
```

By executing the DENY_ALL procedure, you can choose to not trust any database server:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;

Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
--------- ----------------------------------------------
Untrusted All

1 row selected.
```

The ALLOW_SERVER procedure can be used to specify that one particular database is to be trusted:

```
EXECUTE
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER
                          ('SALES.US.AMERICAS.ACME_AUTO.COM');

Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
```

```
--------- ------------------------------------------------
Trusted   SALES.US.AMERICAS.ACME_AUTO.COM

1 row selected.
```

16

DBMS_FGA

The DBMS_FGA package provides fine-grained security functions. Execute privilege on DBMS_FGA is needed for administering audit policies. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only.

> **See Also:** *Oracle9i Application Developer's Guide - Fundamentals* for a fuller discussion and more usage information on DBMS_FGA.

This feature is available for only cost-based optimization. The rule-based optimizer may generate unnecessary audit records since audit monitoring can occur before row filtering. For both the rule-based optimizer and the cost-based optimizer, you can refer to DBA_FGA_AUDIT_TRAIL to analyze the SQL text and corresponding bind variables that are issued.

This chapter discusses the following topics:

Summary of DBMS_FGA Subprograms

Table 16–1 Summary of DBMS_FGA Subprograms

Subprogram	Description
ADD_POLICY Procedure on page 16-2	Creates an audit policy using the supplied predicate as the audit condition
DROP_POLICY Procedure on page 16-3	Drops an audit policy
ENABLE_POLICY Procedure on page 16-4	Enables an audit policy
DISABLE_POLICY Procedure on page 16-5	Disables an audit policy

ADD_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition.

Syntax

```
DBMS_FGA.ADD_POLICY(
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_name     VARCHAR2,
   audit_condition VARCHAR2,
   audit_column    VARCHAR2,
   handler_schema  VARCHAR2,
   /handler_module  VARCHAR2,
   enable          BOOLEAN  );
```

Parameters

Table 16–2 ADD_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited
object_name	The name of the object to be audited
policy_name	The unique name of the policy
audit_condition	A condition in a row that indicates a monitoring condition

Table 16–2 ADD_POLICY Procedure Parameters

Parameter	Description
audit_column	The column to be checked for access. The default is all columns.
handler_schema	The schema that contains the event handler. The default is the current schema.
handler_module	The function name of the event handler; includes the package name if necessary. This is fired only after the first row that matches the audit condition is processed in the query. If the procedure fails with exception, the user SQL statement will fail as well. The default is NULL.
enable	Enables the policy if TRUE, which is the default.

Usage Notes

- An event record will always be inserted into `fga_log$` when the monitored condition becomes TRUE.
- The audit function must have the following interface:
  `PROCEDURE <fname> ( object_schema VARCHAR2, object_name VARCHAR2, policy_name VARCHAR2 ) AS ...`
  `where fname` is the name of the procedure, `schema` is the schema of the table audited, `table` is the table audited, and `policy` is the policy being enforced.
- The audit function is executed as an autonomous transaction.
- Each audit policy is applied to the query individually. That is, as long as the rows being returned fit into any of the audit condition defined on the table, an audit record will be generated, and there will be at most one record generated for each policy.

DROP_POLICY Procedure

This procedure drops an audit policy.

Syntax

```
DBMS_FGA.DROP_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2 );
```

Parameters

Table 16–3 DROP_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited
object_name	The name of the object to be audited
policy_name	The unique name of the policy

Usage Notes

The DBMS_FGA procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_FGA procedures are part of the DDL transaction.

ENABLE_POLICY Procedure

This procedure enables an audit policy.

Syntax

```
DBMS_FGA.ENABLE_POLICY(
   object_schema  VARCHAR2 := NULL,
   object_name    VARCHAR2,
   policy_name    VARCHAR2,
   enable         BOOLEAN := TRUE);
```

Parameters

Table 16–4 ENABLE_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited
object_name	The name of the object to be audited
policy_name	The unique name of the policy
enable	Defaults to TRUE to enable the policy

DISABLE_POLICY Procedure

This procedure disables an audit policy.

Syntax

```
DBMS_FGA.DISABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2 );
```

Parameters

Table 16–5 DISABLE_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited
object_name	The name of the object to be audited
policy_name	The unique name of the policy

17

DBMS_FLASHBACK

Using DBMS_FLASHBACK, you can flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN). When DBMS_FLASHBACK is enabled, the user session uses the Flashback version of the database, and applications can execute against the Flashback version of the database. DBMS_FLASHBACK is automatically turned off when the session ends, either by disconnection or by starting another connection.

PL/SQL cursors opened in Flashback mode return rows as of the flashback time or SCN. Different concurrent sessions (connections) in the database can perform Flashback to different wall-clock times or SCNs. DML and DDL operations and distributed operations are not allowed while a session is running in Flashback mode. You can use PL/SQL cursors opened before disabling Flashback to perform DML.

Under Automatic Undo Management (AUM) mode, you can use retention control to control how far back in time to go for the version of the database you need. If you need to perform a Flashback over a 24-hour period, the DBA should set the undo_retention parameter to 24 hours. This way, the system retains enough undo information to regenerate the older versions of the data.

When enabling Flashback using a wall-clock time, the database chooses an SCN that was generated within five minutes of the time specified. For finer grain control of Flashback, you can enable an SCN. An SCN identifies the exact version of the database. In a Flashback-enabled session, SYSDATE will not be affected; it will continue to provide the current time.

DBMS_FLASHBACK can be used within logon triggers to enable Flashback without changing the application code.

You may want to use DBMS_FLASHBACK for the following reasons:

- Self-service repair. If you accidentally delete rows from a table, you can recover the deleted rows.
- Packaged applications such as e-mail and voicemail. You can use Flashback to restore deleted e-mail by re-inserting the deleted message into the current message box.
- Decision support system (DSS) and online analytical processing (OLAP) applications. You can perform data analysis or data modeling to track seasonal demand, for example.

To use this package, a database administrator must grant `EXECUTE` privileges for `DBMS_FLASHBACK`.

> **See Also:** *Oracle9i Application Developer's Guide - Fundamentals* and *Oracle9i SQL Reference* for detailed information about `DBMS_FLASHBACK`.

This chapter discusses the following topics:

DBMS_FLASHBACK Error Messages

Table 17–1 DBMS_FLASHBACK Error Messages

Error	Description
8182	In Flashback mode, user cannot perform DML or DDL operations.
8184	User cannot enable Flashback within another Flashback session.
8183	User cannot enable Flashback within an uncommitted transaction.
8185	SYS cannot enable Flashback mode. User cannot begin read-only or serializable transactions in Flashback mode.
8180	Time specified is too old.
8181	Invalid system change number specified.

Using DBMS_FLASHBACK: Example

The following example illustrates how Flashback can be used when the deletion of a senior employee triggers the deletion of all the personnel reporting to him. Using the Flashback feature, you can recover and re-insert the missing employees.

```
drop table employee;
drop table keep_scn;

REM keep_scn is a temporary table to store scns that we are interested in

create table keep_scn (scn number);
set echo on
create table employee (
   employee_no   number(5) primary key,
   employee_name varchar2(20),
   employee_mgr  number(5)
      constraint mgr_fkey references employee on delete cascade,
   salary        number,
   hiredate      date
);

REM Populate the company with employees
insert into employee values (1, 'John Doe', null, 1000000, '5-jul-81');
```

```
insert into employee values (10, 'Joe Johnson', 1, 500000, '12-aug-84');
insert into employee values (20, 'Susie Tiger', 10, 250000, '13-dec-90');
insert into employee values (100, 'Scott Tiger', 20, 200000, '3-feb-86');
insert into employee values (200, 'Charles Smith', 100, 150000, '22-mar-88');
insert into employee values (210, 'Jane Johnson', 100, 100000, '11-apr-87');
insert into employee values (220, 'Nancy Doe', 100, 100000, '18-sep-93');
insert into employee values (300, 'Gary Smith', 210, 75000, '4-nov-96');
insert into employee values (310, 'Bob Smith', 210, 65000, '3-may-95');
commit;

REM Show the entire org
select lpad(' ', 2*(level-1)) || employee_name Name
from employee
connect by prior employee_no = employee_mgr
start with employee_no = 1
order by level;

REM Sleep for 5 minutes to avoid querying close to the table creation
REM (the mapping of scn->time has 5 minutes granularity)
execute dbms_lock.sleep(300);

REM Store this snapshot for later access through Flashback
declare
I number;
begin
I := dbms_flashback.get_system_change_number;
insert into keep_scn values (I);
commit;
end;
/

REM Scott decides to retire but the transaction is done incorrectly
delete from employee where employee_name = 'Scott Tiger';
commit;

REM notice that all of scott's employees are gone
select lpad(' ', 2*(level-1)) || employee_name Name
from employee
connect by prior employee_no = employee_mgr
start with employee_no = 1
order by level;

REM Flashback to see Scott's organization
declare
   restore_scn number;
```

```
begin
   select  scn into restore_scn from keep_scn;
   dbms_flashback.enable_at_system_change_number (restore_scn);
end;
/

REM Show Scott's org.
select lpad(' ', 2*(level-1)) || employee_name Name
from employee
connect by prior employee_no = employee_mgr
start with employee_no =
   (select employee_no from employee where employee_name = 'Scott Tiger')
order by level;

REM Restore scott's organization.

declare
   scotts_emp number;
   scotts_mgr number;
   cursor c1 is
      select employee_no, employee_name, employee_mgr, salary, hiredate
      from employee
      connect by prior employee_no = employee_mgr
      start with employee_no =
         (select employee_no from employee where employee_name = 'Scott Tiger');
   c1_rec c1 % ROWTYPE;
begin
   select employee_no, employee_mgr into scotts_emp, scotts_mgr from employee
   where employee_name = 'Scott Tiger';
   /* Open c1 in flashback mode */
   open c1;
   /* Disable Flashback */
   dbms_flashback.disable;
 loop
   fetch c1 into c1_rec;
   exit when c1%NOTFOUND;
   /*
     Note that all the DML operations inside the loop are performed
     with Flashback disabled
   */
   if (c1_rec.employee_mgr = scotts_emp) then
      insert into employee values (c1_rec.employee_no,
         c1_rec.employee_name,
         scotts_mgr,
         c1_rec.salary,
```

```
        c1_rec.hiredate);
   else
   if (c1_rec.employee_no != scotts_emp) then
   insert into employee values (c1_rec.employee_no,
        c1_rec.employee_name,
        c1_rec.employee_mgr,
        c1_rec.salary,
        c1_rec.hiredate);
      end if;
    end if;
 end loop;
end;
/

REM Show the restored organization.
select lpad(' ', 2*(level-1)) || employee_name Name
from employee
connect by prior employee_no = employee_mgr
start with employee_no = 1
order by level;
```

Summary of DBMS_FLASHBACK Subprograms

Table 17–2 DBMS_FLASHBACK Subprograms

Subprogram	Description
ENABLE_AT_TIME Procedure on page 17-7	Enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in `query_time`.
ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure on page 17-7	Takes an SCN as an Oracle number and sets the session snapshot to the specified number. Inside the Flashback mode, all queries will return data consistent as of the specified wall-clock time or SCN.
GET_SYSTEM_CHANGE_NUMBER Function on page 17-8	Returns the current SCN as an Oracle number. You can use the SCN to store specific snapshots.
DISABLE Procedure on page 17-8	Disables the Flashback mode for the entire session.

ENABLE_AT_TIME Procedure

This procedure enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in `query_time`.

Syntax

```
DBMS_FLASHBACK.ENABLE_AT_TIME (
   query_time   IN TIMESTAMP);
```

Parameters

Table 17–3 ENABLE_AT_TIME Procedure Parameters

Parameter	Description
`query_time`	This is an input parameter of type `TIMESTAMP`. A time stamp can be specified in the following ways: Using the TIMESTAMP constructor: Example: `execute dbms_flashback.enable_at_time(TIMESTAMP '2001-01-09 12:31:00')`. Use the Globalization Support (NLS) format and supply a string. The format depends on the Globalization Support settings. Using the `TO_TIMESTAMP` function: Example: `execute dbms_flashback.enable_at_time(TO_TIMESTAMP('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS'))`. You provide the format you want to use. This example shows the `TO_TIMESTAMP` function for February 12, 2001, 2:35 PM. If the time is omitted from query time, it defaults to the beginning of the day, that is, 12:00 A.M. Note that if the query time contains a time zone, the time zone information is truncated.

ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure

This procedure takes an SCN as an input parameter and sets the session snapshot to the specified number.

In the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN.

Syntax

```
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (
```

```
    query_scn IN NUMBER);
```

Parameters

Table 17–4 ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure Parameters

Parameter	Description
query_scn	The system change number (SCN), a version number for the database that is incremented on every transaction commit.

GET_SYSTEM_CHANGE_NUMBER Function

This function returns the current SCN as an Oracle number datatype. You can obtain the current change number and stash it away for later use. This helps you store specific snapshots.

Syntax

```
DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER (
RETURN NUMBER);
```

DISABLE Procedure

This procedure disables the Flashback mode for the entire session.

Syntax

```
DBMS_FLASHBACK.DISABLE;
```

Example

The following example queries the salary of an employee, Joe, on August 30, 2000:

```
EXECUTE dbms_flashback.enable_at_time('30-AUG-2000');
SELECT salary from emp where name = 'Joe'
EXECUTE dbms_flashback.disable;
```

18

DBMS_HS_PASSTHROUGH

The pass-through SQL feature allows an application developer to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows for operations in statements for which there is no equivalent in Oracle.

You can run these statements directly at the non-Oracle system using the PL/SQL package `DBMS_HS_PASSTHROUGH`. Any statement executed with this package is run in the same transaction as regular "transparent" SQL statements.

This chapter discusses the following topics:

Security

The DBMS_HS_PASSTHROUGH package conceptually resides at the non-Oracle system. Procedures and functions in the package must be called by using the appropriate database link to the non-Oracle system.

Summary of DBMS_HS_PASSTHROUGH Subprograms

Table 18–1 DBMS_HS_PASSTHROUGH Package Subprograms

Subprogram	Description
BIND_VARIABLE Procedure on page 18-3	Binds an IN variable positionally with a PL/SQL program variable.
BIND_VARIABLE_RAW Procedure on page 18-4	Binds IN variables of type RAW.
BIND_OUT_VARIABLE Procedure on page 18-5	Binds an OUT variable with a PL/SQL program variable.
BIND_OUT_VARIABLE_RAW Procedure on page 18-7	Binds an OUT variable of datatype RAW with a PL/SQL program variable.
BIND_INOUT_VARIABLE Procedure on page 18-8	Binds IN OUT bind variables.
BIND_INOUT_VARIABLE_ RAW Procedure on page 18-9	Binds IN OUT bind variables of datatype RAW.
CLOSE_CURSOR Procedure on page 18-10	Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system.
EXECUTE_IMMEDIATE Procedure on page 18-11	Runs a (non-SELECT) SQL statement immediately, without bind variables.
EXECUTE_NON_QUERY Function on page 18-12	Runs a (non-SELECT) SQL statement.
FETCH_ROW Function on page 18-13	Fetches rows from a query.
GET_VALUE Procedure on page 18-14	Retrieves column value from SELECT statement, or retrieves OUT bind parameters.
GET_VALUE_RAW Procedure on page 18-15	Similar to GET_VALUE, but for datatype RAW.
OPEN_CURSOR Function on page 18-16	Opens a cursor for running a passthrough SQL statement at the non-Oracle system.

Table 18–1 DBMS_HS_PASSTHROUGH Package Subprograms (Cont.)

Subprogram	Description
PARSE Procedure on page 18-17	Parses SQL statement at non-Oracle system.

BIND_VARIABLE Procedure

This procedure binds an IN variable positionally with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
   c      IN BINARY_INTEGER NOT NULL,
   pos    IN BINARY_INTEGER NOT NULL,
   val    IN <dty>,
   name   IN VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** To bind RAW variables use BIND_VARIABLE_RAW Procedure on page 18-4.

Parameters

Table 18–2 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Value that must be passed to the bind variable name.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–3 BIND_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined: WNDS, RNDS

BIND_VARIABLE_RAW Procedure

This procedure binds IN variables of type RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
   c      IN BINARY_INTEGER NOT NULL,
   pos    IN BINARY_INTEGER NOT NULL,
   val    IN RAW,
   name   IN VARCHAR2);
```

Parameters

Table 18–4 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Value that must be passed to the bind variable.

Table 18–4 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–5 BIND_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_OUT_VARIABLE Procedure

This procedure binds an OUT variable with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c        IN  BINARY_INTEGER NOT NULL,
   pos      IN  BINARY_INTEGER NOT NULL,
   val      OUT <dty>,
   name     IN  VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure on page 18-7.

Parameters

Table 18–6 BIND_OUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–7 BIND_OUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_OUT_VARIABLE_RAW Procedure

This procedure binds an OUT variable of datatype RAW with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW (
   c       IN  BINARY_INTEGER NOT NULL,
   pos     IN  BINARY_INTEGER NOT NULL,
   val     OUT RAW,
   name    IN  VARCHAR2);
```

Parameters

Table 18–8 BIND_OUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.

Table 18–9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

BIND_INOUT_VARIABLE Procedure

This procedure binds IN OUT bind variables.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
   c       IN     BINARY_INTEGER NOT NULL,
   pos     IN     BINARY_INTEGER NOT NULL,
   val     IN OUT <dty>,
   name    IN     VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** For binding IN OUT variables of datatype RAW see BIND_INOUT_VARIABLE_RAW Procedure on page 18-9.

Parameters

Table 18–10 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.

Table 18–10 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–11 BIND_INOUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_INOUT_VARIABLE_RAW Procedure

This procedure binds IN OUT bind variables of datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
   c        IN     BINARY_INTEGER NOT NULL,
   pos      IN     BINARY_INTEGER NOT NULL,
   val      IN OUT RAW,
   name     IN     VARCHAR2);
```

Parameters

Table 18–12 BIND_INOUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed' using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
name	(Optional) Name the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 18–13 BIND_INOUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

CLOSE_CURSOR Procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a "no operation".

Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
   c IN BINARY_INTEGER NOT NULL);
```

Parameters

Table 18–14 CLOSE_CURSOR Procedure Parameters

Parameter	Description
c	Cursor to be released.

Exceptions

Table 18–15 CLOSE_CURSOR Procedure Exceptions

Exception	Description
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

EXECUTE_IMMEDIATE Procedure

This function runs a SQL statement immediately. Any valid SQL command except SELECT can be run immediately. The statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally the SQL statement is run using the PASSTHROUGH SQL protocol sequence of OPEN_CURSOR, PARSE, EXECUTE_NON_QUERY, CLOSE_CURSOR.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
   S IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 18–16 EXECUTE_IMMEDIATE Procedure Parameters

Parameter	Description
s	VARCHAR2 variable with the statement to be executed immediately.

Returns

The number of rows affected by the execution of the SQL statement.

Exceptions

Table 18–17 EXECUTE_IMMEDIATE Procedure Exceptions

Exception	Description
ORA-28551	SQL statement is invalid.
ORA-28544	Max open cursors.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

EXECUTE_NON_QUERY Function

This function runs a SQL statement. The SQL statement cannot be a SELECT statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
   c IN BINARY_INTEGER NOT NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 18–18 EXECUTE_NON_QUERY Function Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.

Returns

The number of rows affected by the SQL statement in the non-Oracle system

Exceptions

Table 18–19 EXECUTE_NON_QUERY Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	BIND_VARIABLE procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

FETCH_ROW Function

This function fetches rows from a result set. The result set is defined with a SQL SELECT statement. When there are no more rows to be fetched, the exception NO_DATA_FOUND is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
   c       IN BINARY_INTEGER NOT NULL,
   first   IN BOOLEAN)
  RETURN BINARY_INTEGER;
```

Parameters

Table 18–20 FETCH_ROW Function Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
first	(Optional) Reexecutes SELECT statement. Possible values: - TRUE: reexecute SELECT statement. - FALSE: fetch the next row, or if run for the first time, then execute and fetch rows (default).

Returns

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 18–21 FETCH_ROW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE Procedure

This procedure has two purposes:

- It retrieves the select list items of SELECT statements, after a row has been fetched.
- It retrieves the OUT bind values, after the SQL statement has been run.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
   c      IN  BINARY_INTEGER NOT NULL,
   pos    IN  BINARY_INTEGER NOT NULL,
   val    OUT <dty>);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** For retrieving values of datatype RAW, see GET_VALUE_RAW Procedure on page 18-15.

Parameters

Table 18–22 GET_VALUE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable or select list item in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 18–23 GET_VALUE Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE_RAW Procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c      IN  BINARY_INTEGER NOT NULL,
   pos    IN  BINARY_INTEGER NOT NULL,
   val    OUT RAW);
```

Parameters

Table 18–24 GET_VALUE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable or select list item in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 18–25 GET_VALUE_RAW Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

OPEN_CURSOR Function

This function opens a cursor for running a pass-through SQL statement at the non-Oracle system. This function must be called for any type of SQL statement

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure CLOSE_CURSOR.

Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
  RETURN BINARY_INTEGER;
```

Returns

The cursor to be used on subsequent procedure and function calls.

Exceptions

Table 18–26 OPEN_CURSOR Function Exceptions

Exception	Description
ORA-28554	Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' OPEN_CURSORS initialization parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

PARSE Procedure

This procedure parses SQL statement at non-Oracle system.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c       IN  BINARY_INTEGER NOT NULL,
   stmt    IN  VARCHAR2       NOT NULL);
```

Parameters

Table 18–27 PARSE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened using function OPEN_CURSOR.
stmt	Statement to be parsed.

Exceptions

Table 18–28 GET_VALUE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28551	SQL statement is illegal.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS, RNDS
```

19

DBMS_IOT

The DBMS_IOT package creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command. DBMS_IOT can also create an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

DBMS_IOT is not loaded during database installation. To install DBMS_IOT, run dbmsiotc.sql and prvtiotc.sql, available in the admin directory.

This chapter discusses the following topics:

- Summary of DBMS_IOT Subprograms

Summary of DBMS_IOT Subprograms

Table 19–1 DBMS_IOT Package Subprograms

Subprogram	Description
BUILD_CHAIN_ROWS_TABLE Procedure on page 19-2	Creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.
BUILD_EXCEPTIONS_TABLE Procedure on page 19-3	Creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

BUILD_CHAIN_ROWS_TABLE Procedure

The BUILD_CHAIN_ROWS_TABLE procedure creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.

Syntax

```
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (
   owner               IN VARCHAR2,
   iot_name            IN VARCHAR2,
   chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```

Parameters

Table 19–2 BUILD_CHAIN_ROWS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
chainrow_table_name	Intended name for the chained-rows table.

Example

```
CREATE TABLE l(a char(16),b char(16), c char(16), d char(240),
PRIMARY KEY(a,b,c)) ORGANIZATION INDEX pctthreshold 10 overflow;
EXECUTE DBMS_IOT.BUILD_CHAIN_ROWS_TABLE('SYS','L','LC');
```

A chained-row table is created with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER_NAME                              VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CLUSTER_NAME                            VARCHAR2(30)
PARTITION_NAME                          VARCHAR2(30)
SUBPARTITION_NAME                       VARCHAR2(30)
HEAD_ROWID                              ROWID
TIMESTAMP                               DATE
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

BUILD_EXCEPTIONS_TABLE Procedure

The BUILD_EXCEPTIONS_TABLE procedure creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

A separate chained-rows table and an exception table should be created for each index-organized table to accommodate its primary key.

> **Note:** This form of chained-rows table and exception table are required only for servers running with Oracle8, Release 8.0 compatibility.

Syntax

```
DBMS_IOT.BUILD_EXCEPTIONS_TABLE (
   owner                IN VARCHAR2,
   iot_name             IN VARCHAR2,
   exceptions_table_name IN VARCHAR2 default 'IOT_EXCEPTIONS');
```

Parameters

Table 19–3 BUILD_EXCEPTIONS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.

Table 19–3 BUILD_EXCEPTIONS_TABLE Procedure Parameters

Parameter	Description
exceptions_table_name	Intended name for exception-table.

Example

```
EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS','L','LE');
```

An exception table for the preceding index-organized table with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
ROW_ID                                  VARCHAR2(30)
OWNER                                   VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CONSTRAINT                              VARCHAR2(30)
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

20

DBMS_JOB

DBMS_JOB subprograms schedule and manage jobs in the job queue.

> **See Also:** For more information on the DBMS_JOB package and the job queue, see *Oracle9i Database Administrator's Guide*

This chapter discusses the following topics:

- Requirements
- Using the DBMS_JOB Package with Oracle Real Application Clusters
- Summary of DBMS_JOB Subprograms

Requirements

There are no database privileges associated with jobs. DBMS_JOB does not allow a user to touch any jobs except their own.

Using the DBMS_JOB Package with Oracle Real Application Clusters

For this example, a constant in DBMS_JOB indicates no mapping among jobs and instances; that is, jobs can be executed by any instance.

DBMS_JOB.SUBMIT

To submit a job to the job queue, use the following syntax:

```
DBMS_JOB.SUBMIT( JOB OUT BINARY_INTEGER,
WHAT     IN VARCHAR2, NEXT_DATE IN DATE DEFAULTSYSDATE,
INTERVAL IN VARCHAR2 DEFAULT 'NULL',
NO_PARSE IN BOOLEAN DEFAULT FALSE,
INSTANCE IN BINARY_INTEGER DEFAULT ANY_INSTANCE,
FORCE    IN BOOLEAN DEFAULT FALSE);
```

Use the parameters INSTANCE and FORCE to control job and instance affinity. The default value of INSTANCE is 0 (zero) to indicate that any instance can execute the job. To run the job on a certain instance, specify the INSTANCE value. Oracle displays error ORA-23319 if the INSTANCE value is a negative number or NULL.

The FORCE parameter defaults to FALSE. If force is TRUE, any positive integer is acceptable as the job instance. If FORCE is FALSE, the specified instance must be running, or Oracle displays error number ORA-23428.

DBMS_JOB.INSTANCE

To assign a particular instance to execute a job, use the following syntax:

```
DBMS_JOB.INSTANCE(  JOB IN BINARY_INTEGER,
INSTANCE                IN BINARY_INTEGER,
FORCE                   IN BOOLEAN DEFAULT FALSE);
```

The FORCE parameter in this example defaults to FALSE. If the instance value is 0 (zero), job affinity is altered and any available instance can execute the job despite the value of force. If the INSTANCE value is positive and the FORCE parameter is FALSE, job affinity is altered only if the specified instance is running, or Oracle displays error ORA-23428.

If the FORCE parameter is TRUE, any positive integer is acceptable as the job instance and the job affinity is altered. Oracle displays error ORA-23319 if the INSTANCE value is negative or NULL.

DBMS_JOB.CHANGE

To alter user-definable parameters associated with a job, use the following syntax:

```
DBMS_JOB.CHANGE(  JOB IN BINARY_INTEGER,
WHAT                  IN VARCHAR2 DEFAULT NULL,
NEXT_DATE             IN DATE DEFAULT NULL,
INTERVAL              IN VARCHAR2 DEFAULT NULL,
INSTANCE              IN BINARY_INTEGER DEFAULT NULL,
FORCE                 IN BOOLEAN DEFAULT FALSE );
```

Two parameters, INSTANCE and FORCE, appear in this example. The default value of INSTANCE is NULL indicating that job affinity will not change.

The default value of FORCE is FALSE. Oracle displays error ORA-23428 if the specified instance is not running and error ORA-23319 if the INSTANCE number is negative.

DBMS_JOB.RUN

The FORCE parameter for DBMS_JOB.RUN defaults to FALSE. If force is TRUE, instance affinity is irrelevant for running jobs in the foreground process. If force is FALSE, the job can run in the foreground only in the specified instance. Oracle displays error ORA-23428 if force is FALSE and the connected instance is the incorrect instance.

```
DBMS_JOB.RUN(
   JOB    IN BINARY_INTEGER,
   FORCE  IN BOOLEAN DEFAULT FALSE);
```

> **See Also:** *Oracle9i Real Application Clusters Concepts* for more information

Summary of DBMS_JOB Subprograms

Table 20–1 DBMS_JOB Package Subprograms

Subprogram	Description
SUBMIT Procedure on page 20-4	Submits a new job to the job queue.

Table 20–1 DBMS_JOB Package Subprograms (Cont.)

Subprogram	Description
REMOVE Procedure on page 20-6	Removes specified job from the job queue.
CHANGE Procedure on page 20-6	Alters any of the user-definable parameters associated with a job.
WHAT Procedure on page 20-7	Alters the job description for a specified job.
NEXT_DATE Procedure on page 20-8	Alters the next execution time for a specified job.
INSTANCE Procedure on page 20-8	Assigns a job to be run by a instance.
INTERVAL Procedure on page 20-9	Alters the interval between executions for a specified job.
BROKEN Procedure on page 20-10	Disables job execution.
RUN Procedure on page 20-11	Forces a specified job to run.
USER_EXPORT Procedure on page 20-11	Re-creates a given job for export.
USER_EXPORT Procedure on page 20-12	Re-creates a given job for export with instance affinity.

SUBMIT Procedure

This procedure submits a new job. It chooses the job from the sequence `sys.jobseq`.

Syntax

```
DBMS_JOB.SUBMIT (
   job       OUT BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE DEFAULT sysdate,
   interval  IN  VARCHAR2 DEFAULT 'null',
   no_parse  IN  BOOLEAN DEFAULT FALSE,
   instance  IN  BINARY_INTEGER DEFAULT any_instance,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–2 SUBMIT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Next date when the job will be run.
interval	Date function that calculates the next time to run the job. The default is NULL. This must evaluate to a either a future point in time or NULL.
no_parse	A flag. The default is FALSE. If this is set to FALSE, then Oracle parses the procedure associated with the job. If this is set to TRUE, then Oracle parses the procedure associated with the job the first time that the job is run. For example, if you want to submit a job before you have created the tables associated with the job, then set this to TRUE.
instance	When a job is submitted, specifies which instance can run the job.
force	If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

Example

This submits a new job to the job queue. The job calls the procedure DBMS_DDL.ANALYZE_OBJECT to generate optimizer statistics for the table DQUON.ACCOUNTS. The statistics are based on a sample of half the rows of the ACCOUNTS table. The job is run every 24 hours:

```
VARIABLE jobno number;
BEGIN
   DBMS_JOB.SUBMIT(:jobno,
      'dbms_ddl.analyze_object(''TABLE'',
      ''DQUON'', ''ACCOUNTS'',
```

```
      ''ESTIMATE'', NULL, 50);'
      SYSDATE, 'SYSDATE + 1');
   commit;
END;
/
Statement processed.
print jobno
JOBNO
----------
14144
```

REMOVE Procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

Syntax

```
DBMS_JOB.REMOVE (
   job       IN  BINARY_INTEGER );
```

Parameters

Table 20–3 REMOVE Procedure Parameters

Parameter	Description
job	Number of the job being run.

Example

```
EXECUTE DBMS_JOB.REMOVE(14144);
```

CHANGE Procedure

This procedure changes any of the user-settable fields in a job.

Syntax

```
DBMS_JOB.CHANGE (
   job       IN  BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE,
   interval  IN  VARCHAR2,
   instance  IN  BINARY_INTEGER DEFAULT NULL,
```

```
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–4 CHANGE Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Date of the next refresh.
interval	Date function; evaluated immediately before the job starts running.
instance	When a job is submitted, specifies which instance can run the job. This defaults to NULL, which indicates that instance affinity is not changed.
force	If this is FALSE, then the specified instance (to which the instance number change) must be running. Otherwise, the routine raises an exception. If this is TRUE, then any positive integer is acceptable as the job instance.

Usage Notes

The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

If the parameters `what`, `next_date`, or `interval` are `NULL`, then leave that value as it is.

Example

```
EXECUTE DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
```

WHAT Procedure

This procedure changes what an existing job does, and replaces its environment.

Syntax

```
DBMS_JOB.WHAT (
   job       IN  BINARY_INTEGER,
```

```
   what      IN  VARCHAR2);
```

Parameters

Table 20–5 WHAT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.

Some legal values of what (assuming the routines exist) are:

- 'myproc(''10-JAN-82'', next_date, broken);'
- 'scott.emppackage.give_raise(''JENKINS'', 30000.00);'
- 'dbms_job.remove(job);'

NEXT_DATE Procedure

This procedure changes when an existing job next runs.

Syntax

```
DBMS_JOB.NEXT_DATE (
   job       IN  BINARY_INTEGER,
   next_date IN  DATE);
```

Parameters

Table 20–6 NEXT_DATE Procedure Parameters

Parameter	Description
job	Number of the job being run.
next_date	Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.

INSTANCE Procedure

This procedure changes job instance affinity.

Syntax

```
DBMS_JOB.INSTANCE (
   job        IN BINARY_INTEGER,
   instance   IN BINARY_INTEGER,
   force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–7 INSTANCE Procedure Parameters

Parameter	Description
job	Number of the job being run.
instance	When a job is submitted, a user can specify which instance can run the job.
force	If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.

INTERVAL Procedure

This procedure changes how often a job runs.

Syntax

```
DBMS_JOB.INTERVAL (
   job       IN  BINARY_INTEGER,
   interval  IN  VARCHAR2);
```

Parameters

Table 20–8 INTERVAL Procedure Parameters

Parameter	Description
job	Number of the job being run.
interval	Date function, evaluated immediately before the job starts running.

Usage Notes

If the job completes successfully, then this new date is placed in next_date. interval is evaluated by plugging it into the statement select interval into next_date from dual;

The interval parameter must evaluate to a time in the future. Legal intervals include:

Interval	Description
'sysdate + 7'	Run once a week.
'next_day(sysdate,''TUESDAY'')'	Run once every Tuesday.
'null'	Run only once.

If interval evaluates to NULL and if a job completes successfully, then the job is automatically deleted from the queue.

BROKEN Procedure

This procedure sets the broken flag. Broken jobs are never run.

Syntax

```
DBMS_JOB.BROKEN (
   job       IN  BINARY_INTEGER,
   broken    IN  BOOLEAN,
   next_date IN  DATE DEFAULT SYSDATE);
```

Parameters

Table 20–9 Broken Procedure Parameters

Parameter	Description
job	Number of the job being run.
broken	Job broken: IN value is FALSE.
next_data	Date of the next refresh.

Note: If you set job as broken while it is running, Oracle resets the job's status to normal after the job completes. Therefore, only execute this procedure for jobs that are not running.

RUN Procedure

This procedure runs job JOB now. It runs it even if it is broken.

Running the job recomputes next_date. See view user_jobs.

Syntax

```
DBMS_JOB.RUN (
   job       IN  BINARY_INTEGER,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–10 Run Procedure Parameters

Parameter	Description
job	Number of the job being run.
force	If this is TRUE, then instance affinity is irrelevant for running jobs in the foreground process. If this is FALSE, then the job can be run in the foreground only in the specified instance.

Example

```
EXECUTE DBMS_JOB.RUN(14144);
```

Caution: This reinitializes the current session's packages.

Exceptions

An exception is raised if force is FALSE, and if the connected instance is the wrong one.

USER_EXPORT Procedure

This procedure produces the text of a call to re-create the given job.

Syntax

```
DBMS_JOB.USER_EXPORT (
   job    IN     BINARY_INTEGER,
   mycall IN OUT VARCHAR2);
```

Parameters

Table 20–11 USER_EXPORT Procedure Parameter

Parameter	Description
job	Number of the job being run.
mycall	Text of a call to recreate the given job.

USER_EXPORT Procedure

This procedure alters instance affinity (8*i* and after) and preserves the compatibility.

Syntax

```
DBMS_JOB.USER_EXPORT (
   job      IN     BINARY_INTEGER,
   mycall   IN OUT VARCHAR2,
   myinst   IN OUT VARCHAR2);
```

Parameters

Table 20–12 USER_EXPORT Procedure Parameters

Parameter	Description
job	Number of the job being run.
mycall	Text of a call to re-create a given job.
myinst	Text of a call to alter instance affinity.

21

DBMS_LDAP

DBMS_LDAP provides functions and procedures to access data from LDAP servers. To use DBMS_LDAP, you must first load it into the database. Use the catldap.sql script located in the $ORACLE_HOME/rdbms/admin directory.

> **See Also:** *Oracle Internet Directory Application Developer's Guide* for more information on using DBMS_LDAP.

This chapter discusses the following topics:

Exception Summary

Table 21–1 lists the exceptions generated by DBMS_LDAP.

Table 21–1 DBMS_LDAP Exception Summary

Exception Name	Oracle Error	Cause of Exception
general_error	31202	Raised anytime an error is encountered that does not have a specific PL/SQL exception associated with it. The error string contains the description of the problem in the local language of the user.
init_failed	31203	Raised by DBMS_LDAP.init if there are some problems.
invalid_session	31204	Raised by all functions and procedures in the DBMS_LDAP package if they are passed an invalid session handle.
invalid_auth_method	31205	Raised by DBMS_LDAP.bind_s if the authentication method requested is not supported.
invalid_search_scope	31206	Raised by all of the search functions if the scope of the search is invalid.
invalid_search_time_val	31207	Raised by time based search function: DBMS_LDAP.search_st if it is given an invalid value for the time limit.
invalid_message	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.
count_entry_error	31209	Raised by DBMS_LDAP.count_entries if it cannot count the entries in a given result set.
get_dn_error	31210	Raised by DBMS_LDAP.get_dn if the DN of the entry it is retrieving is NULL.
invalid_entry_dn	31211	Raised by all the functions that modify/add/rename an entry if they are presented with an invalid entry DN.
invalid_mod_array	31212	Raised by all functions that take a modification array as an argument if they are given an invalid modification array.
invalid_mod_option	31213	Raised by DBMS_LDAP.populate_mod_array if the modification option given is anything other than MOD_ADD, MOD_DELETE or MOD_REPLACE.
invalid_mod_type	31214	Raised by DBMS_LDAP.populate_mod_array if the attribute type that is being modified is NULL.
invalid_mod_value	31215	Raised by DBMS_LDAP.populate_mod_array if the modification value parameter for a given attribute is NULL.
invalid_rdn	31216	Raised by all functions and procedures that expect a valid RDN if the value of the RDN is NULL.

Table 21–1 DBMS_LDAP Exception Summary

Exception Name	Oracle Error	Cause of Exception
invalid_ newparent	31217	Raised by DBMS_LDAP.rename_s if the new parent of an entry being renamed is NULL.
invalid_ deleteoldrdn	31218	Raised by DBMS_LDAP.rename_s if the deleteoldrdn parameter is invalid.
invalid_ notypes	31219	Raised by DBMS_LDAP.explode_dn if the notypes parameter is invalid.
invalid_ssl_ wallet_loc	31220	Raised by DBMS_LDAP.open_ssl if the wallet location is NULL but the SSL authentication mode requires a valid wallet.
invalid_ssl_ wallet_ password	31221	Raised by DBMS_LDAP.open_ssl if the wallet password given is NULL.
invalid_ssl_ auth_mode	31222	Raised by DBMS_LDAP.open_ssl if the SSL authentication mode is not one of 1, 2, or 3.
mts_mode_not_ supported	31398	Raised by the functions init, bind_s or simple_bind_s if they are ever invoked in MTS mode.

Summary of Data Types

The DBMS_LDAP package uses the data types shown in Table 21–2.

Table 21–2 DBMS_LDAP Summary of Data Types

Data-Type	Purpose
SESSION	Holds the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.
MESSAGE	Holds a handle to the message retrieved from the result set. This is used by all functions that work with entries, attributes, and values.
MOD_ARRAY	Holds a handle into the array of modifications being passed into either modify_s or add_s.
TIMEVAL	Passes time limit information to the LDAP API functions that require a time limit.
BER_ELEMENT	Holds a handle to a BER structure used for decoding incoming messages.

Table 21–2 DBMS_LDAP Summary of Data Types

Data-Type	Purpose
STRING_COLLECTION	Holds a list of VARCHAR2 strings which can be passed on to the LDAP server.
BINVAL_COLLECTION	Holds a list of RAW data which represent binary data.
BERVAL_COLLECTION	Holds a list of BERVAL values that are used for populating a modification array.

Summary of DBMS_LDAP Subprograms

Table 21–3 DBMS_LDAP Subprograms

Function or Procedure	Description
init Function on page 21-6	Initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.
simple_bind_s Function on page 21-7	Performs simple username / password based authentication to the directory server.
bind_s Function on page 21-9	Performs complex authentication to the directory server.
unbind_s Function on page 21-10	Closes an active LDAP session.
compare_s Function on page 21-11	Tests if a particular attribute in a particular entry has a particular value.
search_s Function on page 21-13	Performs a synchronous search in the LDAP server. It returns control to the PL / SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the server.
search_st Function on page 21-15	Performs a synchonous search in the LDAP server with a client side timeout. It returns control to the PL / SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the client or the server.
first_entry Function on page 21-17	Retrieves the first entry in the result set returned by either search_s or search_st.
next_entry Function on page 21-18	Iterates to the next entry in the result set of a search operation.

Table 21–3 DBMS_LDAP Subprograms (Cont.)

Function or Procedure	Description
count_entries Function on page 21-20	Counts the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions `first_entry` and `next_entry`.
first_attribute Function on page 21-21	Fetches the first attribute of a given entry in the result set.
next_attribute Function on page 21-22	Fetches the next attribute of a given entry in the result set.
get_dn Function on page 21-24	Retrieves the X.500 distinguished name of given entry in the result set.
get_values Function on page 21-25	Retrieves all of the values associated for a given attribute in a given entry.
get_values_len Function on page 21-26	Retrieves values of attributes that have a Binary syntax.
delete_s Function on page 21-28	Removes a leaf entry in the LDAP Directory Information Tree.
modrdn2_s Function on page 21-29	Renames the relative distinguished name of an entry.
err2string Function on page 21-30	Converts an LDAP error code to string in the local language in which the API is operating.
create_mod_array Function on page 21-31	Allocates memory for array modification entries that are applied to an entry using the `modify_s` functions.
populate_mod_array (String Version) Procedure on page 21-32	Populates one set of attribute information for add or modify operations.
populate_mod_array (Binary Version) Procedure on page 21-34	Populates one set of attribute information for add or modify operations. This procedure call has to happen after `DBMS_LDAP.create_mod_array` is called.
modify_s Function on page 21-35	Performs a sychronous modification of an existing LDAP directory entry.
add_s Function on page 21-37	Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, we have to call `DBMS_LDAP.creat_mod_array` and `DBMS_LDAP.populate_mod_array` first.
free_mod_array Procedure on page 21-38	Frees the memory allocated by `DBMS_LDAP.create_mod_array`.

Table 21–3 DBMS_LDAP Subprograms (Cont.)

Function or Procedure	Description
count_values Function on page 21-39	Counts the number of values returned by `DBMS_LDAP.get_values`.
count_values_len Function on page 21-40	Counts the number of values returned by DBMS_LDAP.get_values_len.
rename_s Function on page 21-41	Renames an LDAP entry synchronously.
explode_dn Function on page 21-43	Breaks a DN up into its components.
open_ssl Function on page 21-44	Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

init Function

This function initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.

Syntax

```
DBMS_LDAP.init (
   hostname  IN VARCHAR2,
   portnum   IN PLS_INTEGER )
RETURN SESSION;
```

Parameters

Table 21–4 init Function Parameters

Parameter	Description
`hostname (IN)`	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server. Each host name in the list may include a port number, which is separated from the host with a colon (:). The hosts are tried in the order listed, stopping with the first one to which a successful connection is made.
`portnum (IN)`	Contains the TCP port number to connect to. If a host includes a port number, this parameter is ignored. If this parameter is not specified and the host name does not contain the port number, the default port number 389 is assumed.

Return Values

Table 21–5 init Function Return Values

Value	Description
SESSION	A handle to an LDAP session that can be used for further calls into the API.

Exceptions

Table 21–6 init Function Exceptions

Exception	Description
init_failed	Raised when there is a problem contacting the LDAP server.
ts_mode_not_ supported	Raised if DBMS_LDAP.init is invoked from a user session that is logged onto the database using an MTS service.
general_error	For all other errors. The error string associated with the exception describes the error in detail.

Usage Notes

DBMS_LDAP.init is the first function that should be called in order to establish a session to the LDAP server. DBMS_LDAP.init returns a session handle, a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. This routine returns NULL and raises the INIT_FAILED exception if the session cannot be initialized. Subsequent to the call to init, the connection must be authenticated using DBMS_LDAP.bind_s or DBMS_LDAP.simple_bind_s.

See Also:

- "simple_bind_s Function" on page 21-7
- "bind_s Function" on page 21-9

simple_bind_s Function

This function can be used to perform simple username/password based authentication to the directory server.

Syntax

```
DBMS_LDAP.simple_bind_s (
   ld      IN SESSION,
   dn      IN VARCHAR2,
   passwd  IN VARCHAR2)
RETURN PLS_INTEGER;
```

Parameters

Table 21–7 simple_bind_s Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.
`dn (IN)`	The distinguished name of the user under which you are trying to login.
`passwd (IN)`	A text string containing the password.

Return Values

Table 21–8 simple_bind_s Function Return Values

Value	Description
`PLS_INTEGER`	`DBMS_LDAP.SUCCESS` on a successful completion. If there was a problem, one of the exceptions in Table 21–9 is raised.

Exceptions

Table 21–9 simple_bind_s Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle ld is invalid.
`mts_mode_not_supported`	Raised if `DBMS_LDAP.init` is invoked from a user session that is logged onto as an MTS service.
`general_error`	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

DBMS_LDAP.simple_bind_s can be used to authenticate a user whose directory distinguished name and directory password are known. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init.

bind_s Function

This function performs complex authentication to the directory server.

Syntax

```
DBMS_LDAP.bind_s (
   ld    IN SESSION,
   dn    IN VARCHAR2,
   cred  IN VARCHAR2,
   meth  IN PLS_INTEGER )
RETURN PLS_INTEGER;
```

Parameters

Table 21–10 bind_s Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The distinguished name of the user under which you are trying to login.
cred	A text string containing the credentials used for authentication.
meth	The authentication method.

Return Values

Table 21–11 bind_s Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on a successful completion. One of the exceptions in Table 21–12 is raised if there is a problem.

Exceptions

Table 21–12 bind_s Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle ld is invalid.
`invalid_auth_method`	Raised if the authentication method requested is not supported.
`mts_mode_not_supported`	Raised if invoked from a user session that is logged onto an MTS service.
`general_error`	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

`DBMS_LDAP.bind_s` can be used to authenticate a user. It can be called only after a valid LDAP session handle is obtained from a call to `DBMS_LDAP.init`.

See Also:

- "init Function" on page 21-6
- "simple_bind_s Function" on page 21-7

unbind_s Function

This function closes an active LDAP session.

Syntax

```
DBMS_LDAP.unbind_s (
   ld  IN SESSION )
RETURN PLS_INTEGER;
```

Parameters

Table 21–13 unbind_s Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.

Return Values

Table 21–14 unbind_s Function Return Values

Value	Description
PLS_INTEGER	SUCCESS on proper completion. One of the exceptions listed in Table 21–15 is raised otherwise.

Exceptions

Table 21–15 unbind_s Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The `unbind_s` function sends an unbind request to the server, closes all open connections associated with the LDAP session, and disposes of all resources associated with the session handle before returning. After a call to this function, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld.

See Also:

- "simple_bind_s Function" on page 21-7
- "bind_s Function" on page 21-9

compare_s Function

This function tests whether a particular attribute in a particular entry has a particular value.

Syntax

```
DBMS_LDAP.compare_s (
   ld     IN SESSION,
   dn     IN VARCHAR2,
   attr   IN VARCHAR2,
   value  IN VARCHAR2)
```

```
RETURN PLS_INTEGER;
```

Parameters

Table 21–16 compare_s Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle
dn (IN)	The name of the entry to compare against
attr (IN)	The attribute to compare against.
value (IN)	A string attribute value to compare against

Return Values

Table 21–17 compare_s Function Return Values

Value	Description
PLS_INTEGER	COMPARE_TRUE is the given attribute that has a matching value. COMPARE_FALSE if the value of the attribute does not match the value given.

Exceptions

Table 21–18 compare_s Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function compare_s can be used to assert if the value of a given attribute stored in the directory server matches a certain value.This operation can only be performed on attributes whose syntax definition allows them to be compared. The compare_s function can only be called after a valid LDAP session handle has been obtained from the init function and authenticated using the bind_s or simple_bind_s functions.

See Also: "bind_s Function" on page 21-9.

search_s Function

This function performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the server.

Syntax

```
FUNCTION search_s  (
   ld        IN SESSION,
   base      IN VARCHAR2,
   scope     IN PLS_INTEGER,
   filter    IN VARCHAR2,
   attrs     IN STRING_COLLECTION,
   attronly  IN PLS_INTEGER,
   res       OUT MESSAGE)
RETURN PLS_INTEGER;
```

Parameters

Table 21–19 search_s Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle.
base (IN)	The dn of the entry at which to start the search.
scope (IN)	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter (IN)	A character string representing the search filter. The value NULL can be passed to indicate that the filter (objectclass=*) which matches all entries is to be used.
attrs (IN)	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS (1.1) can be used as the only string in the array to indicate that no attribute types are returned by the server. The special constant string ALL_USER_ATTRS (*) can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are returned.

Table 21–19 search_s Function Parameters

Parameter	Description
attrsonly (IN)	A boolean value that must be zero if both attribute types and values are returned, and nonzero if only types are wanted.
res (OUT)	This is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 21–20 search_s Function Return Value

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res (OUT parameter)	If the search succeeded and there are entries, this parameter is set to a nonnull value that can be used to iterate through the result set.

Exceptions

Table 21–21 search_s Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL, or SCOPE_SUBTREE.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

This function issues a search operation, and does not return control to the user environment until all of the results have been returned from the server. Entries returned from the search, if any, are contained in the res parameter. This parameter is opaque to the caller. Entries, attributes, values, and so on can be extracted by calling the parsing routines described in the following sections.

See Also:

- "search_st Function" on page 21-15
- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18

search_st Function

This function performs a synchronous search in the LDAP server with a client-side timeout. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the client or the server.

Syntax

```
DBMS_LDAP.search_st (
   ld        IN  SESSION,
   base      IN  VARCHAR2,
   scope     IN  PLS_INTEGER,
   filter    IN  VARCHAR2,
   attrs     IN  STRING_COLLECTION,
   attronly  IN  PLS_INTEGER,
   tv        IN  TIMEVAL,
   res       OUT MESSAGE)
RETURN PLS_INTEGER;
```

Parameters

Table 21–22 search_st Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.
`base (IN)`	The dn of the entry at which to start the search.
`scope (IN)`	One of `SCOPE_BASE (0x00)`, `SCOPE_ONELEVEL (0x01)`, or `SCOPE_SUBTREE (0x02)`, indicating the scope of the search.
`filter (IN)`	A character string representing the search filter. The value `NULL` can be passed to indicate that the filter (objectclass=*) which matches all entries is to be used.

Table 21–22 search_st Function Parameters

Parameter	Description
attrs (IN)	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS (1.1) can be used as the only string in the array to indicate that no attribute types are returned by the server. The special constant string ALL_USER_ATTRS (*) can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are returned.
attrsonly (IN)	A boolean value that must be zero if both attribute types and values are returned, and nonzero if only types are wanted.
tv (IN)	The timeout value expressed in seconds and microseconds that should be used for this search.
res (OUT)	This is a result parameter that will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 21–23 search_st Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res (OUT parameter)	If the search succeeded and there are entries, this parameter is set to a NON_NULL value that can be used to iterate through the result set.

Exceptions

Table 21–24 search_st Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_value	Raised if the time value specified for the timeout is invalid.

Table 21–24 search_st Function Exceptions

Exception	Description
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

This function is very similar to DBMS_LDAP.search_s, except that it requires a timeout value.

See Also:

- "search_s Function" on page 21-13
- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18

first_entry Function

This function retrieves the first entry in the result set returned by either search_s or search_st

Syntax

```
DBMS_LDAP.first_entry (
   ld   IN SESSION,
   msg  IN MESSAGE )
RETURN MESSAGE;
```

Parameters

Table 21–25 first_entry Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle.
msg (IN)	The search result obtained by a call to one of the synchronous search routines.

Return Values

Table 21–26 first_entry Return Values

Value	Description
MESSAGE	A handle to the first entry in the list of entries returned from the LDAP server. It is set to NULL if there was an error and an exception is raised.

Exceptions

Table 21–27 first_entry Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The function `first_entry` should always be the first function used to retrieve the results from a search operation.

See Also:

- "next_entry Function" on page 21-18
- "search_s Function" on page 21-13
- "search_st Function" on page 21-15

next_entry Function

This function iterates to the next entry in the result set of a search operation.

Syntax

```
DBMS_LDAP.next_entry (
   ld   IN SESSION,
   msg  IN MESSAGE )
RETURN MESSAGE;
```

Parameters

Table 21–28 next_entry Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.
`msg (IN)`	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 21–29 next_entry Function Return Values

Value	Description
MESSAGE	A handle to the next entry in the list of entries returned from the LDAP server. It is set to `NULL` if there was an error and an exception is raised.

Exceptions

Table 21–30 next_entry Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle, ld is invalid.
`invalid_message`	Raised if the incoming msg handle is invalid.

Usage Notes

The function `next_entry` should always be called after a call to `first_entry`. Also, the return value of a successful call to `next_entry` should be used as `msg` argument used in a subsequent call to `next_entry` to fetch the next entry in the list.

See Also:

- "search_s Function" on page 21-13
- "search_st Function" on page 21-15
- "first_entry Function" on page 21-17

count_entries Function

This function counts the number of entries in the result set. It can also count the number of entries remaining during a traversal of the result set using a combination of the functions `first_entry` and `next_entry`.

Syntax

```
DBMS_LDAP.count_entries (
   ld   IN SESSION,
   msg  IN MESSAGE )
RETURN PLS_INTEGER;
```

Parameters

Table 21–31 count_entry Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle
`msg (IN)`	The search result, as obtained by a call to one of the synchronous search routines

Return Values

Table 21–32 count_entry Function Return Values

Value	Description
PLS INTEGER	Nonzero if there are entries in the result set -1 if there was a problem.

Exceptions

Table 21–33 count_entry Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle ld is invalid.
`invalid_message`	Raised if the incoming msg handle is invalid.
`count_entry_error`	Raised if there was a problem in counting the entries.

Usage Notes

The `count_entries` function returns the number of entries contained in a chain of entries. If an error occurs, such as the `res` parameter being invalid, `-1` is returned. The `count_entries` call can also be used to count the number of entries that remain in a chain if called with a message, entry, or reference returned by `first_message`, `next_message`, `first_entry`, `next_entry`, `first_reference`, and `next_reference`.

See Also:

- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18

first_attribute Function

This function fetches the first attribute of a given entry in the result set.

Syntax

```
DBMS_LDAP.first_attribute (
   ld        IN  SESSION,
   msg       IN  MESSAGE,
   ber_elem  OUT BER_ELEMENT)
RETURN VARCHAR2;
```

Parameters

Table 21–34 first_attribute Function Parameter

Parameter	Description
`ld (IN)`	A valid LDAP session handle
`msg (IN)`	The entry whose attributes are to be stepped through, as returned by `first_entry` or `next_entry`
`ber_elem (OUT)`	A handle to a `BER ELEMENT` that is used to keep track of which attribute in the entry has been read

Return Values

Table 21–35 first_attribute Function Return Values

Value	Description
VARCHAR2	The name of the attribute if it exists. NULL if no attribute exists or if an error occurred.
ber_elem	A handle used by DBMS_LDAP.next_attribute to iterate over all of the attributes

Exceptions

Table 21–36 first_attribute Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute should be used in the next call to next_attribute to iterate through the various attributes of an entry. The name of the attribute returned from a call to first_attribute can in turn be used in calls to the functions get_values or get_values_len to get the values of that particular attribute.

See Also:
- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18
- "next_attribute Function" on page 21-22
- "get_values Function" on page 21-25
- "get_values_len Function" on page 21-26

next_attribute Function

This function fetches the next attribute of a given entry in the result set.

Syntax

```
DBMS_LDAP.next_attribute (
```

```
   ld        IN SESSION,
   msg       IN MESSAGE,
   ber_elem  IN BER_ELEMENT)
RETURN VARCHAR2;
```

Parameters

Table 21–37 next_attribute Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle.
msg (IN)	The entry whose attributes are to be stepped through, as returned by first_entry or next_entry.
ber_elem (IN)	A handle to a BER ELEMENT that is used to keep track of which attribute in the entry has been read.

Return Values

Table 21–38 next_attribute Function Return Values

Value	Description
VARCHAR2	The name of the attribute, if it exists.

Exceptions

Table 21–39 next_attribute Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute should be used in the next call to next_attribute to iterate through the various attributes of an entry. The name of the attribute returned from a call to next_attribute can in turn be used in calls to get_values or get_values_len to get the values of that particular attribute.

See Also:

- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18
- "first_attribute Function" on page 21-21
- "get_values Function" on page 21-25
- "get_values_len Function" on page 21-26

get_dn Function

This function retrieves the X.500 distinguished name of a given entry in the result set.

The function `first_attribute` fetches the first attribute of a given entry in the result set

Syntax

```
DBMS_LDAP.get_dn (
   ld   IN SESSION,
   msg  IN MESSAGE)
RETURN VARCHAR2;
```

Parameters

Table 21–40 get_dn Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.
`msg (IN)`	The entry whose DN is to be returned.

Return Values

Table 21–41 get_dn Function Return Values

Value	Description
VARCHAR2	The X.500 distinguished name of the entry as a PL/SQL string. NULL if there was a problem.

Exceptions

Table 21–42 get_dn Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN.

Usage Notes

The function get_dn can be used to retrieve the DN of an entry as the program logic is iterating through the result set. This be used as an input to explode_dn to retrieve the individual components of the DN.

See Also: "explode_dn Function" on page 21-43.

get_values Function

This function retrieves all of the values associated for a given attribute in a given entry.

Syntax

```
DBMS_LDAP.get_values (
   ld       IN SESSION,
   ldapentry  IN MESSAGE,
   attr       IN VARCHAR2)
RETURN STRING_COLLECTION;
```

Parameters

Table 21–43 get_values Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle.
ldapentry (IN)	A valid handle to an entry returned from a search result.
attr (IN)	The name of the attribute for which values are being sought.

Return Values

Table 21–44 get_values Function Return Values

Value	Description
STRING_COLLECTION	A PL/SQL string collection containing all of the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 21–45 get_values Function Exceptions

Exception	Description
invalid session	Raised if the session handle ld is invalid.
invalid message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values can only be called after the handle to entry has been first retrieved by a call to either first_entry or next_entry. The name of the attribute can be known beforehand, and it can also be determined by a call to first_attribute or next_attribute. The function get_values always assumes that the datatype of the attribute it is retrieving is String. For retrieving binary datatypes, use get_values_len.

See Also:

- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18
- "get_values_len Function" on page 21-26
- "count_values Function" on page 21-39

get_values_len Function

This function retrieves values of attributes that have a Binary syntax.

Syntax

```
DBMS_LDAP.get_values_len (
   ld         IN SESSION,
```

```
   ldapentry  IN MESSAGE,
   attr       IN VARCHAR2)
RETURN BINVAL_COLLECTION;
```

Parameters

Table 21–46 get_values_len Function Parameters

Parameter	Description
`ld (IN)`	A valid LDAP session handle.
`ldapentrymsg (IN)`	A valid handle to an entry returned from a search result.
`attr (IN)`	The string name of the attribute for which values are being sought.

Return Values

Table 21–47 get_values_len Function Return Values

Value	Description
`BINVAL_COLLECTION`	A PL/SQL Raw collection containing all the values of the given attribute. `NULL` if there are no values associated with the given attribute.

Exceptions

Table 21–48 get_values_len Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle ld is invalid.
`invalid_message`	Raised if the incoming entry handle is invalid

Usage Notes

The function `get_values_len` can only be called after the handle to entry has been retrieved by a call to either `first_entry` or `next_entry`. The name of the attribute can be known beforehand, and it can also be determined by a call to `first_attribute` or `next_attribute`. This function can be used to retrieve both binary and nonbinary attribute values.

See Also:

- "first_entry Function" on page 21-17
- "next_entry Function" on page 21-18
- "get_values Function" on page 21-25
- "count_values_len Function" on page 21-40

delete_s Function

This function removes a leaf entry in the LDAP Directory Information Tree.

Syntax

```
DBMS_LDAP.delete_s (
   ld       IN SESSION,
   entrydn  IN VARCHAR2)
RETURN PLS_INTEGER;
```

Parameters

Table 21–49 delete_s Function Parameters

Parameter Name	Description
`ld (IN)`	A valid LDAP session
`entrydn (IN)`	The X.500 distinguished name of the entry to delete.

Return Values

Table 21–50 delete_s Function Return Values

Value	Description
`PLS_INTEGER`	`DBMS_LDAP.SUCCESS` if the delete operation wa successful. An exception is raised otherwise.

Exceptions

Table 21–51 delete_s Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle ld is invalid.

Table 21–51 delete_s Function Exceptions

Exception	Description
invalid_entry_dn	Raised if the distinguished name of the entry is invalid
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function delete_s can be used to remove only leaf level entries in the LDAP DIT. A leaf level entry is an entry that does not have any children / LDAP entries under it. It cannot be used to delete nonleaf entries.

See Also: "modrdn2_s Function" on page 21-29.

modrdn2_s Function

This function modrdn2_s can be used to rename the relative distinguished name of an entry.

Syntax

```
DBMS_LDAP.modrdn2_s (
   ld           IN SESSION,
   entrydn      IN VARCHAR2
   newrdn       IN VARCHAR2
   deleteoldrdn IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 21–52 modrdn2_s Function Parameters

Parameter	Description
ld (IN)	A valid LDAP session handle.
entrydn (IN)	The distinguished name of the entry. (This entry must be a leaf node in the DIT.).
newrdn (IN)	The new relative distinguished name of the entry.
deleteoldrdn (IN)	A boolean value that if nonzero, indicates that the attribute values from the old name should be removed from the entry.

Return Values

Table 21–53 modrdn2_s Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.

Exceptions

Table 21–54 modrdn2_s Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
invalid_rdn	Invalid LDAP RDN.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.
general error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

This function can be used to rename the leaf nodes of a DIT. It simply changes the relative distinguished name by which they are known. The use of this function is being deprecated in the LDAP v3 standard. Please use `rename_s`, which can achieve the same foundation.

See Also: "rename_s Function" on page 21-41.

err2string Function

This function converts an LDAP error code to string in the local language in which the API is operating

Syntax

```
DBMS_LDAP.err2string (
   ldap_err  IN PLS_INTEGER )
RETURN VARCHAR2;
```

Parameters

Table 21–55 err2string Function Parameters

Parameter	Description
ldap_err (IN)	An error number returned from one the API calls.

Return Values

Table 21–56 err2string Function Return Values

Value	Description
VARCHAR2	A character string appropriately translated to the local language which describes the error in detail.

Exceptions

Table 21–57 err2string Function Exceptions

Exception	Description
N/A	None.

Usage Notes

In this release, the exception handling mechanism automatically invokes this if any of the API calls encounter an error.

create_mod_array Function

This function allocates memory for array modification entries that are applied to an entry using the modify_s or add_s functions.

Syntax

```
DBMS_LDAP.create_mod_array (
   num  IN PLS_INTEGER)
RETURN MOD_ARRAY;
```

Parameters

Table 21–58 create_mod_array Function Parameters

Parameter	Description
num (IN)	The number of the attributes that you want to add or modify.

Return Values

Table 21–59 create_mod_array Function Return Values

Value	Description
MOD_ARRAY	The data structure holds a pointer to an LDAP mod array. NULL if there was a problem.

Exceptions

Table 21–60 create_mod_array Function Exceptions

Exception	Description
N/A	No LDAP specific exception is raised

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is required to call DBMS_LDAP.free_mod_array to free memory after the calls to add_s or modify_s have completed.

See Also:

- "populate_mod_array (String Version) Procedure" on page 21-32
- "modify_s Function" on page 21-35
- "add_s Function" on page 21-37
- "free_mod_array Procedure" on page 21-38

populate_mod_array (String Version) Procedure

This procedure populates one set of attribute information for add or modify operations.

Syntax

```
DBMS_LDAP.populate_mod_array (
   modptr    IN DBMS_LDAP.MOD_ARRAY,
   mod_op    IN PLS_INTEGER,
   mod_type  IN VARCHAR2,
   modval    IN DBMS_LDAP.STRING_COLLECTION);
```

Parameters

Table 21–61 populate_mod_array (String Version) Procedure Parameters

Parameter	Description
`modptr (IN)`	The data structure holds a pointer to an LDAP mod array.
`Mod_op (IN)`	This field specifies the type of modification to perform.
`Mod_type (IN)`	This field indicates the name of the attribute type to which the modification applies.
`Modval (IN)`	This field specifies the attribute values to add, delete, or replace. It is for the string values only.

Return Values

Table 21–62 populate_mod_array (String Version) Procedure Return Values

Value	Description
N/A	-

Exceptions

Table 21–63 populate_mod_array (String Version) Procedure Exceptions

Exception	Description
`invalid_mod_array`	Invalid LDAP mod array.
`invalid_mod_option`	Invalid LDAP mod option.
`invalid_mod_type`	Invalid LDAP mod type.
`invalid_mod_value`	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It has to happen after DBMS_LDAP.create_mod_array is called.

See Also:

- "create_mod_array Function" on page 21-31
- "modify_s Function" on page 21-35
- "add_s Function" on page 21-37
- "free_mod_array Procedure" on page 21-38

populate_mod_array (Binary Version) Procedure

This procedure populates one set of attribute information for add or modify operations. This procedure call has to happen after DBMS_LDAP.create_mod_array is called.

Syntax

```
PROCEDURE populate_mod_array
    (modptr   IN DBMS_LDAP.MOD_ARRAY,
     mod_op   IN PLS_INTEGER,
     mod_type IN VARCHAR2,
     modval   IN DBMS_LDAP.BERVAL_COLLECTION);
```

Parameters

Table 21–64 populate_mod_array (Binary Version) Procedure Parameters

Parameter	Description
modptr (IN)	The data structure holds a pointer to an LDAP mod array.
Mod_op (IN)	This field specifies the type of modification to perform.
Mod_typ (IN)	This field indicates the name of the attribute type to which the modification applies.
Modval (IN)	This field specifies the attribute values to add, delete, or replace. It is for the binary values.

Return Values

Table 21–65 populate_mod_array (Binary Version) Procedure Return Values

Value	Description
N / A	-

Exceptions

Table 21–66 populate_mod_array (Binary Version) Procedure Exceptions

Exception	Description
`invalid_mod_array`	Invalid LDAP mod array.
`invalid_mod_option`	Invalid LDAP mod option.
`invalid_mod_type`	Invalid LDAP mod type.
`invalid_mod_value`	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It has to happen after `DBMS_LDAP.create_mod_array` is called.

See Also:

- "create_mod_array Function" on page 21-31
- "modify_s Function" on page 21-35
- "add_s Function" on page 21-37
- "free_mod_array Procedure" on page 21-38

modify_s Function

This function performs a synchronous modification of an existing LDAP directory entry.

Syntax

```
DBMS_LDAP.modify_s (
   ld       IN DBMS_LDAP.SESSION,
   entrydn  IN VARCHAR2,
```

```
   modptr   IN DBMS_LDAP.MOD_ARRAY)
RETURN PLS_INTEGER;
```

Parameters

Table 21–67 modify_s Function Parameters

Parameter	Description
ld (IN)	A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.
entrydn (IN)	Specifies the name of the directory entry whose contents are to be modified.
modptr (IN)	The handle to an LDAP mod structure, as returned by a successful call to DBMS_LDAP.create_mod_array.

Return Values

Table 21–68 modify_s Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the modification operation

Exceptions

Table 21–69 modify_s Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

This function call has to follow successful calls of DBMS_LDAP.create_mod_array and DBMS_LDAP.populate_mod_array.

See Also:

- "create_mod_array Function" on page 21-31
- "populate_mod_array (String Version) Procedure" on page 21-32
- "add_s Function" on page 21-37
- "free_mod_array Procedure" on page 21-38

add_s Function

This function adds a new entry to the LDAP directory sychronously. Before calling `add_s`, you must call `DBMS_LDAP.create_mod_array` and `DBMS_LDAP.populate_mod_array`.

Syntax

```
DBMS_LDAP.add_s (
   ld       IN DBMS_LDAP.SESSION,
   entrydn  IN VARCHAR2,
   modptr   IN DBMS_LDAP.MOD_ARRAY)
RETURN PLS_INTEGER;
```

Parameters

Table 21–70 add_s Function Parameters

Parameter	Description
`ld (IN)`	A handle to an LDAP session, as returned by a successful call to `DBMS_LDAP.init`.
`Entrydn (IN)`	Specifies the name of the directory entry to be created.
`Modptr (IN)`	The handle to an LDAP mod structure, as returned by successful call to `DBMS_LDAP.create_mod_array`.

Return Values

Table 21–71 add_s Function Return Values

Value	Description
`PLS_INTEGER`	The indication of the success or failure of the modification operation.

Exceptions

Table 21–72 add_s Function Exceptions

Exception	Description
`invalid_session`	Invalid LDAP session.
`invalid_entry_dn`	Invalid LDAP entry dn.
`invalid_mod_array`	Invalid LDAP mod array.

Usage Notes

The parent entry of the entry to be added must already exist in the directory. This function call has to follow successful calls of `DBMS_LDAP.create_mod_array` and `DBMS_LDAP.populate_mod_array`.

See Also:

- "create_mod_array Function" on page 21-31
- "populate_mod_array (String Version) Procedure" on page 21-32
- "modify_s Function" on page 21-35
- "free_mod_array Procedure" on page 21-38

free_mod_array Procedure

This procedure frees the memory allocated by `DBMS_LDAP.create_mod_array`.

Syntax

```
DBMS_LDAP.free_mod_array (
   modptr  IN DBMS_LDAP.MOD_ARRAY);
```

Parameters

Table 21–73 free_mod_array Procedure Parameters

Parameter	Description
`modptr (in)`	The handle to an LDAP mod structure, as returned by successful call to `DBMS_LDAP.create_mod_array`.

Return Values

Table 21–74 free_mod_array Procedure Return Value

Value	Description
N / A	-

Exceptions

Table 21–75 free_mod_array Procedure Exceptions

Exception	Description
N / A	No LDAP specific exception is raised.

See Also:

- "create_mod_array Function" on page 21-31
- "populate_mod_array (String Version) Procedure" on page 21-32
- "modify_s Function" on page 21-35
- "add_s Function" on page 21-37

count_values Function

This function counts the number of values returned by `DBMS_LDAP.get_values`.

Syntax

```
DBMS_LDAP.count_values (
   values  IN DBMS_LDAP.STRING_COLLECTION)
RETURN PLS_INTEGER;
```

Parameters

Table 21–76 count_values Function Parameters

Parameter	Description
`values (IN)`	The collection of string values.

Return Values

Table 21–77 count_values Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 21–78 count_values Function Exceptions

Exception	Description
N/A	No LDAP specific exception is raised.

See Also:

- "get_values Function" on page 21-25
- "count_values_len Function" on page 21-40

count_values_len Function

This function counts the number of values returned by `DBMS_LDAP.get_values_len`.

Syntax

```
DBMS_LDAP.count_values_len (
   values  IN DBMS_LDAP.BINVAL_COLLECTION)
RETURN PLS_INTEGER;
```

Parameters

Table 21–79 count_values_len Function Parameters

Parameter	Description
values (IN)	The collection of binary values.

Return Values

Table 21–80 count_values_len Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 21–81 count_values_len Function Exceptions

Exception	Description
N / A	No LDAP specific exception is raised.

See Also:

- "get_values_len Function" on page 21-26
- "count_values Function" on page 21-39

rename_s Function

This function renames an LDAP entry synchronously.

Syntax

```
DBMS_LDAP.rename_s (
   ld            IN SESSION,
   dn            IN VARCHAR2,
   newrdn        IN VARCHAR2,
   newparent     IN VARCHAR2,
   deleteoldrdn  IN PLS_INTEGER,
   serverctrls   IN LDAPCONTROL,
   clientctrls   IN LDAPCONTROL)
RETURN PLS_INTEGER;
```

Parameters

Table 21–82 rename_s Function Parameters

Parameter	Description
ld (IN)	A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.
Dn (IN)	Specifies the name of the directory entry to be renamed or moved.
newrdn (IN)	Specifies the new RDN.
Newparent (IN)	Specifies the DN of the new parent.
Deleteoldrdn (IN)	Specifies if the old RDN should be retained. If this value is 1, then the old RDN is removed.
Serverctrls (IN)	Currently not supported.
Clientctrls (IN)	Currently not supported.

Return Values

Table 21–83 rename_s Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 21–84 rename_s Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_entry_dn	Invalid LDAP DN.
invalid_rdn	Invalid LDAP RDN.
invalid_newparent	Invalid LDAP newparent.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.

See Also: "modrdn2_s Function" on page 21-29.

explode_dn Function

This function breaks a DN up into its components.

Syntax

```
DBMS_LDAP.explode_dn (
   dn       IN VARCHAR2,
   notypes  IN PLS_INTEGER)
RETURN STRING_COLLECTION;
```

Parameters

Table 21–85 explode_dn Function Parameters

Parameter	Description
`dn (IN)`	Specifies the name of the directory entry to be broken up.
`Notypes (IN)`	Specifies if the attribute tags will be returned. If this value is not 0, no attribute tags are returned.

Return Values

Table 21–86 explode_dn Function Return Values

Value	Description
`STRING_COLLECTION`	An array of strings. If the DN cannot be broken up, `NULL` is returned.

Exceptions

Table 21–87 explode_dn Function Exceptions

Exception	Description
`invalid_entry_dn`	Invalid LDAP DN.
`invalid_notypes`	Invalid LDAP notypes value.

See Also: "get_dn Function" on page 21-24.

open_ssl Function

This function establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Syntax

```
DBMS_LDAP.open_ssl (
   ld               IN SESSION,
   sslwrl           IN VARCHAR2,
   sslwalletpasswd  IN VARCHAR2,
   sslauth          IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 21–88 open_ssl Function Parameters

Parameter	Description
ld (IN)	A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.
Sslwrl (IN)	Specifies the wallet location (Required for one-way or two-way SSL connection.)
sslwalletpasswd (IN)	Specifies the wallet password (Required for one-way or two-way SSL connection.)
sslauth (IN)	Specifies the SSL Authentication Mode (1 for no authentication required, 2 for one way authentication required, 3 for two way authentication required.

Return Values

Table 21–89 open_ssl Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 21–90 open_ssl Function Exceptions

Exception	Description
`invalid_session`	Invalid LDAP Session.
`invalid_ssl_wallet_loc`	Invalid LDAP SSL wallet location.
`invalid_ssl_wallet_passwd`	Invalid LDAP SSL wallet passwd.
`invalid_ssl_auth_mode`	Invalid LDAP SSL authentication mode.

Usage Notes

Call `DBMS_LDAP.init` first to acquire a valid LDAP session.

See Also: "init Function" on page 21-6.

22

DBMS_LIBCACHE

DBMS_LIBCACHE prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution. The value of compiling the cache of an instance is to prepare the information the application requires to execute in advance of failover or switchover.

Compiling a shared cursor consists of open, parse, and bind operations, plus the type-checking and execution plan functions performed at the first execution. All of these steps are executed in advance by the package DBMS_LIBCACHE for SELECT statements. The open and parse functions are executed in advance for PL/SQL and DML. For PL/SQL, executing the parse phase has the effect of loading all library cache heaps other than the MCODE.

This chapter discusses the following topics:

Requirements

To execute DBMS_LIBCACHE you must directly access the same objects as do SQL statements. You can best accomplish this by utilizing the same user id as the original system on the remote system. When there are multiple schema users, DBMS_LIBCACHE should be called for each. Alternately, DBMS_LIBCACHE may be called with the generic user PARSER. However, this user cannot parse the SQL that uses objects with access granted though roles. This is a standard PL/SQL security limitation.

Summary of DBMS_LIBCACHE Subprograms

Table 22–1 DBMS_SESSION Subprograms

Subprogram	Description
COMPILE_CURSORS_FROM_ REMOTE Procedure on page 22-2	Extracts SQL in batch from the source instance and compiles the SQL at the target instance.

COMPILE_CURSORS_FROM_REMOTE Procedure

This procedure extracts SQL in batch from the source instance and compiles the SQL at the target instance.

Syntax

```
DBMS_LIBCACHE.COMPILE_CURSORS_FROM_REMOTE('LIBC_LINK', {MY_USER}, 1,
1024000);
```

Parameters

Table 22–2 COMPILE_CURSORS_FROM_REMOTE Procedure Parameters

Parameter	Description
Database Link Name	The database link pointing to the instance used for extracting the SQL statements.
Source username	Parsing username for the SQL statements extracted.
Execution threshold	Lower bound on the number of executions. Below this value cursors will not be selected for compiling.

Table 22–2 COMPILE_CURSORS_FROM_REMOTE Procedure Parameters

Parameter	Description
`Sharable memory threshold`	The lower bound for the size of the shared memory consumed by the context area on the source instance. Below this value cursors will not be selected for compiling.

Usage Notes

Note the following:

- You must provide a `Database link name` and a Source user name as these are mandatory parameters. The syntax demonstrates the addition of the two optional parameters for preparsing all SQL larger than 1MB.
- `Database link name` - The connection may use either a password file or an LDAP authorization. A default database link, `libc_link`, is created when the catalog program, `catlibc.sql`, is executed. There is no actual default value as this parameter is mandatory for releases with `dbms_libcache$def.ACCESS_METHOD = DB_LINK_METHOD.`
- `Source user name` - This parameter allows the package to be executed in the matching local parsing user id. When using this parameter it is usual to be connected to the same username locally. If the username is supplied it must be a valid value. The name is not case sensitive.
- `Execution threshold` - The execution count on a cursor value is reset whenever the cursor is reloaded. This parameter allows the application to extract and compile statements with executions for example, greater than 3. The default value is 1. This means SQL statements that have never executed, including invalid SQL statements, will not be extracted.
- `Sharable memory threshold` - This parameter allows the application to extract and compile statements with shared memory for example, greater than 1024000 bytes. With the default value (1000), you can skip cursors that are invalid and so are never executed.

23

DBMS_LOB

The DBMS_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use DBMS_LOB to access and manipulation specific parts of a LOB or complete LOBs.

This package must be created under SYS (connect internal). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

DBMS_LOB can read and modify BLOBs, CLOBs, and NCLOBs; it provides read-only operations for BFILEs. The bulk of the LOB operations are provided by this package.

> **See Also:** *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

This chapter discusses the following topics:

LOB Locators for DBMS_LOB

All DBMS_LOB subprograms work based on LOB locators. For the successful completion of DBMS_LOB subprograms, you must provide an input locator that represents a LOB that already exists in the database tablespaces or external file system. See also Chapter 1 of *Oracle9i Application Developer's Guide - Large Objects (LOBs).*

To use LOBs in your database, you must first use SQL data definition language (DDL) to define the tables that contain LOB columns.

Internal LOBs

To populate your table with internal LOBs after LOB columns are defined in a table, you use the SQL data manipulation language (DML) to initialize or populate the locators in the LOB columns.

External LOBs

For an external LOB to be represented by a LOB locator, you must:

- Ensure that a DIRECTORY object representing a valid, existing physical directory has been defined, and that physical files (the LOBs you plan to add) exist with read permission for Oracle. If your operating system uses case-sensitive path names, then be sure you specify the directory in the correct format.
- Pass the DIRECTORY object and the filename of the external LOB you are adding to the BFILENAME( ) function to create a LOB locator for your external LOB.

Once you have completed these tasks, you can insert or update a row containing a LOB column using the given LOB locator.

After the LOBs are defined and created, you can then SELECT from a LOB locator into a local PL/SQL LOB variable and use this variable as an input parameter to DBMS_LOB for access to the LOB value.

For details on the different ways to do this, you must refer to the section of the *Oracle9i Application Developer's Guide - Large Objects (LOBs)* that describes *Accessing External LOBs (BFILEs).*

Temporary LOBs

For temporary LOBs, you must use the OCI, PL/SQL, or another programmatic interface to create or manipulate them. Temporary LOBs can be either BLOBs, CLOBs, or NCLOBs.

Datatypes, Constants, and Exceptions for DBMS_LOB

Datatypes

Parameters for the DBMS_LOB subprograms use these datatypes:

Table 23–1 DBMS_LOB Datatypes

Type	Description
BLOB	A source or destination binary LOB.
RAW	A source or destination RAW buffer (used with BLOB).
CLOB	A source or destination character LOB (including NCLOB).
VARCHAR2	A source or destination character buffer (used with CLOB and NCLOB).
INTEGER	Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.
BFILE	A large, binary object stored outside the database.

The DBMS_LOB package defines no special types. NCLOB is a special case of CLOBs for fixed-width and varying-width, multibyte national character sets. The clause ANY_CS in the specification of DBMS_LOB subprograms for CLOBs enables them to accept a CLOB or NCLOB locator variable as input.

Constants

DBMS_LOB defines the following constants:

```
file_readonly CONSTANT BINARY_INTEGER := 0;
lob_readonly  CONSTANT BINARY_INTEGER := 0;
lob_readwrite CONSTANT BINARY_INTEGER := 1;
lobmaxsize    CONSTANT INTEGER        := 4294967295;
call          CONSTANT PLS_INTEGER    := 12;
session       CONSTANT PLS_INTEGER    := 10;
```

Oracle supports a maximum LOB size of 4 gigabytes (2^{32}). However, the amount and offset parameters of the package can have values between 1 and 4294967295 (2^{32}-1).

The PL/SQL 3.0 language specifies that the maximum size of a RAW or VARCHAR2 variable is 32767 *bytes*.

> **Note:** The value 32767 bytes is represented by maxbufsize in the following sections.

Exceptions

Table 23–2 DBMS_LOB Exceptions

Exception	Code	Description
invalid_argval	21560	The argument is expecting a nonNULL, valid value but the argument value passed in is NULL, invalid, or out of range.
access_error	22925	You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.
noexist_directory	22285	The directory leading to the file does not exist.
nopriv_directory	22286	The user does not have the necessary access privileges on the directory alias or the file for the operation.
invalid_directory	22287	The directory alias used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.
operation_failed	22288	The operation attempted on the file failed.
unopened_file	22289	The file is not open for the required operation to be performed.
open_toomany	22290	The number of open files has reached the maximum limit.

Security for DBMS_LOB

Any DBMS_LOB subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any DBMS_LOB subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

With Oracle8*i*, when creating the procedure, users can set the AUTHID to indicate whether they want definer's rights or invoker's rights. For example:

```
CREATE PROCEDURE proc1 authid definer ...
```

or

```
CREATE PROCEDURE proc1 authid current_user ....
```

> **See Also:** For more information on AUTHID and privileges, see *PL/SQL User's Guide and Reference*

You can provide secure access to BFILEs using the DIRECTORY feature discussed in BFILENAME function in the *Oracle9i Application Developer's Guide - Large Objects (LOBs)* and the *Oracle9i SQL Reference*.

Rules and Limitations for DBMS_LOB

- The following rules apply in the specification of subprograms in this package:
  - length and offset parameters for subprograms operating on BLOBs and BFILEs must be specified in terms of *bytes*.
  - length and offset parameters for subprograms operating on CLOBs must be specified in terms of *characters*.
  - offset and amount parameters are always in *characters* for CLOBs / NCLOBs and in *bytes* for BLOBs / BFILEs.
- A subprogram raises an INVALID_ARGVAL exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of LOB data are permitted: Negative offsets from the tail of the LOB are not permitted.
  2. Only positive, nonzero values are permitted for the parameters that represent size and positional quantities, such as amount, offset, newlen, nth, and so on. Negative offsets and ranges observed in Oracle SQL string functions and operators are not permitted.
  3. The value of offset, amount, newlen, nth must not exceed the value lobmaxsize (4GB-1) in any DBMS_LOB subprogram.
  4. For CLOBs consisting of fixed-width multibyte characters, the maximum value for these parameters must not exceed (lobmaxsize / character_width_in_bytes) characters.

     For example, if the CLOB consists of 2-byte characters, such as:

```
JA16SJISFIXED
```

Then, the maximum `amount` value should not exceed:

```
4294967295/2 = 2147483647 characters.
```

- PL/SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for `RAW` and `VARCHAR2` parameters used in `DBMS_LOB` subprograms. For example, if you declare a variable to be:

  ```
  charbuf VARCHAR2(3000)
  ```

  Then, `charbuf` can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for `DBMS_LOB` subprograms for `CLOBs` and `NCLOBs`.

- The `%CHARSET` clause indicates that the form of the parameter with `%CHARSET` must match the form of the `ANY_CS` parameter to which it refers.

  For example, in `DBMS_LOB` subprograms that take a `VARCHAR2` buffer parameter, the form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. If the input `LOB` parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input `LOB` parameter is of type `CLOB`, then the buffer must contain `CHAR` data.

  For `DBMS_LOB` subprograms that take two `CLOB` parameters, both `CLOB` parameters must have the same form; that is, they must both be `NCLOBs`, or they must both be `CLOBs`.

- If the value of `amount` plus the `offset` exceeds 4 GB (that is, `lobmaxsize`+1) for `BLOBs` and `BFILEs`, and (`lobmaxsize/character_width_in_bytes`)+1 for `CLOBs` in calls to update subprograms (that is, `APPEND`, `COPY`, `TRIM`, `WRITE` and `WRITEAPPEND` subprograms), then access exceptions are raised.

  Under these input conditions, read subprograms, such as `READ`, `COMPARE`, `INSTR`, and `SUBSTR`, read until `End of Lob/File` is reached. For example, for a `READ` operation on a `BLOB` or `BFILE`, if the user specifies `offset` value of 3 GB and an `amount` value of 2 GB, then `READ` reads only ((`4GB-1`)`-3GB`) bytes.

- Functions with `NULL` or invalid input values for parameters return a `NULL`. Procedures with `NULL` values for destination `LOB` parameters raise exceptions.

- Operations involving patterns as parameters, such as `COMPARE`, `INSTR`, and `SUBSTR` do not support regular expressions or special matching characters (such as `%` in the `LIKE` operator in SQL) in the `pattern` parameter or substrings.

- The End Of LOB condition is indicated by the READ procedure using a NO_DATA_FOUND exception. This exception is raised only upon an attempt by the user to read beyond the end of the LOB/FILE. The READ buffer for the last read contains 0 bytes.
- For consistent LOB updates, you must lock the row containing the destination LOB before making a call to any of the procedures (mutators) that modify LOB data.
- Unless otherwise stated, the default value for an offset parameter is 1, which indicates the first byte in the BLOB or BFILE data, and the first character in the CLOB or NCLOB value. No default values are specified for the amount parameter — you must input the values explicitly.
- You must lock the row containing the destination internal LOB before calling any subprograms that modify the LOB, such as APPEND, COPY, ERASE, TRIM, or WRITE. These subprograms do not implicitly lock the row containing the LOB.

BFILE-Specific Rules and Limitations

- The subprograms COMPARE, INSTR, READ, SUBSTR, FILECLOSE, FILECLOSEALL and LOADFROMFILE operate only on an *opened* BFILE locator; that is, a successful FILEOPEN call must precede a call to any of these subprograms.
- For the functions FILEEXISTS, FILEGETNAME and GETLENGTH, a file's open / close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the DIRECTORY object and the file.
- DBMS_LOB does not support any concurrency control mechanism for BFILE operations.
- In the event of several open files in the session whose closure has not been handled properly, you can use the FILECLOSEALL subprogram to close all files opened in the session and resume file operations from the beginning.
- If you are the creator of a DIRECTORY, or if you have system privileges, then use the CREATE OR REPLACE, DROP, and REVOKE statements in SQL with extreme caution.

  If you, or other grantees of a particular directory object, have several open files in a session, then any of the preceding commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to invoke a program or anonymous block that calls FILECLOSEALL, reopen your files, and restart your file operations.

- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after *both* normal and abnormal termination of operations on the BFILE.

  In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than SESSION_MAX_OPEN_FILES.

  In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an exception occurs, only the exception handler has access to the BFILE variable in its most current state.

  After the exception transfers program control outside the PL/SQL program block, all references to the open BFILEs are lost. The result is a larger open file count which may or may not exceed the SESSION_MAX_OPEN_FILES value.

  For example, consider a READ operation past the end of the BFILE value, which generates a NO_DATA_FOUND exception:

```
DECLARE
       fil BFILE;
       pos INTEGER;
       amt BINARY_INTEGER;
       buf RAW(40);
BEGIN
       SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
       dbms_lob.open(fil, dbms_lob.lob_readonly);
       amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
       dbms_lob.read(fil, amt, pos, buf);
       dbms_output.put_line('Read F1 past EOF: '||
           utl_raw.cast_to_varchar2(buf));
       dbms_lob.close(fil);
END;

ORA-01403: no data found
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10
```

  After the exception has occurred, the BFILE locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```
DECLARE
     fil BFILE;
```

```
     pos INTEGER;
     amt BINARY_INTEGER;
     buf RAW(40);
BEGIN
     SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
     dbms_lob.open(fil, dbms_lob.lob_readonly);
     amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
     dbms_lob.read(fil, amt, pos, buf);
     dbms_output.put_line('Read F1 past EOF: '||
          utl_raw.cast_to_varchar2(buf));
     dbms_lob.close(fil);
     exception
     WHEN no_data_found
     THEN
       BEGIN
         dbms_output.put_line('End of File reached. Closing file');
         dbms_lob.fileclose(fil);
         -- or dbms_lob.filecloseall if appropriate
       END;
END;
     /

Statement processed.
End of File reached. Closing file
```

In general, you should ensure that files opened in a PL/SQL block using DBMS_LOB are closed before normal or abnormal termination of the block.

Temporary LOBs

Oracle8*i* supports the definition, creation, deletion, access, and update of temporary LOBs. Your temporary tablespace stores the temporary LOB data. Temporary LOBs are not permanently stored in the database. Their purpose is mainly to perform transformations on LOB data.

A temporary LOB is empty when it is created. By default, all temporary LOBs are deleted at the end of the session in which they were created. If a process dies unexpectedly or if the database crashes, then temporary LOBs are deleted, and the space for temporary LOBs is freed.

In Oracle8*i*, there is also an interface to let you group temporary LOBs together into a logical bucket. The duration represents this logical store for temporary LOBs. Each temporary LOB can have separate storage characteristics, such as CACHE/NOCACHE. There is a default store for every session into which temporary LOBs are placed if

you don't specify a specific duration. Additionally, you are able to perform a free operation on durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or transaction management for temporary LOBs. Because CR and rollbacks are not supported for temporary LOBs, you must free the temporary LOB and start over again if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is potentially a performance impact if you assign multiple locators to the same temporary LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while another locator is also pointing to it. The locator on which a modification was performed now points to a new copy of the temporary LOB. Other locators no longer see the same data as the locator through which the modification was made. A deep copy was not incurred by permanent LOBs in these types of situations, because CR snapshots and version pages enable users to see their own versions of the LOB cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having multiple pointers to locators point to the same temporary LOB locator, if necessary. In PL/SQL, you must avoid using more than one locator for each temporary LOB. The temporary LOB locator can be passed by reference to other procedures.

Because temporary LOBs are not associated with any table schema, there are no meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary LOB instance by a user causes the engine to create and return a locator to the LOB data. The PL/SQL DBMS_LOB package, PRO*C, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server.

Temporary LOBs do not support the EMPTY_BLOB or EMPTY_CLOB functions that are supported for permanent LOBs. The EMPTY_BLOB function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the DBMS_LOB package by using the appropriate FREETEMPORARY or OCIDurationEnd statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and DBMS_LOB statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or DBMS_LOB COPY command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

Oracle keeps track of temporary LOBs for each session in a v$ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist for each session. V$ views are for DBA use. From the session, Oracle can determine which user owns the temporary LOBs. By using V$TEMPORARY_LOBS in conjunction with DBA_SEGMENTS, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

Temporary LOBs Usage Notes

1. All functions in DBMS_LOB return NULL if any of the input parameters are NULL. All procedures in DBMS_LOB raise an exception if the LOB locator is input as NULL.
2. Operations based on CLOBs do not verify if the character set IDs of the parameters (CLOB parameters, VARCHAR2 buffers and patterns, and so on) match. It is the user's responsibility to ensure this.
3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.
4. Temporary LOBs still adhere to value semantics in order to be consistent with permanent LOBs and to try to conform to the ANSI standard for LOBs. As a result, each time a user does an OCILobLocatatorAssign, or the equivalent assignment in PL/SQL, the database makes a copy of the temporary LOB.

   Each locator points to its own LOB value. If one locator is used to create a temporary LOB, and then is assigned to another LOB locator using OCILobLOcatorAssign in OCI or through an assignment operation in

PL/SQL, then the database copies the original temporary LOB and causes the second locator to point to the copy.

In order for users to modify the same LOB, they must go through the same locator. In OCI, this can be accomplished fairly easily by using pointers to locators and assigning the pointers to point to the same locator. In PL/SQL, the same LOB variable must be used to update the LOB to get this effect.

The following example shows a place where a user incurs a copy, or at least an extra roundtrip to the server.

```
DECLARE
  a blob;
  b blob;
BEGIN
  dbms_lob.createtemporary(b, TRUE);
  -- the following assignment results in a deep copy
  a := b;
END;
```

The PL/SQL compiler makes temporary copies of actual arguments bound to OUT or IN OUT parameters. If the actual parameter is a temporary LOB, then the temporary copy is a deep (value) copy.

The following PL/SQL block illustrates the case where the user incurs a deep copy by passing a temporary LOB as an IN OUT parameter.

```
DECLARE
  a blob;
  procedure foo(parm IN OUT blob) is
  BEGIN
   ...
  END;
BEGIN
  dbms_lob.createtemporary(a, TRUE);
  -- the following call results in a deep copy of the blob a
  foo(a);
END;
```

To minimize deep copies on PL/SQL parameter passing, use the NOCOPY compiler hint where possible.

The duration parameter passed to dbms_lob.createtemporary() is a hint. The duration of the new temp LOB is the same as the duration of the locator variable in PL/SQL. For example, in the preceding program block, the program

variable a has the duration of the residing frame. Therefore at the end of the block, memory of a will be freed at the end of the function.

If a PL/SQL package variable is used to create a temp LOB, it will have the duration of the package variable, which has a duration of SESSION.

```
BEGIN
   y clob;
  END;
/
BEGIN
   dbms_lob.createtemporary(package.y, TRUE);
END;
```

See Also: . *PL/SQL User's Guide and Reference* for more information on NOCOPY syntax

Exceptions

Table 23–3 DBMS_LOB Exceptions

Exception	Code	Description
INVALID_ARGVAL	21560	Value for argument %s is not valid.
ACCESS_ERROR	22925	Attempt to read or write beyond maximum LOB size on %s.
NO_DATA_FOUND		EndofLob indicator for looping read operations. This is not a hard error.
VALUE_ERROR	6502	PL/SQL error for invalid values to subprogram's parameters.

Summary of DBMS_LOB Subprograms

Table 23–4 DBMS_LOB Subprograms

Subprogram	Description
APPEND Procedure on page 23-15	Appends the contents of the source LOB to the destination LOB.
CLOSE Procedure on page 23-17	Closes a previously opened internal or external LOB.
COMPARE Function on page 23-18	Compares two entire LOBs or parts of two LOBs.

Table 23–4 DBMS_LOB Subprograms (Cont.)

Subprogram	Description
COPY Procedure on page 23-21	Copies all, or part, of the source LOB to the destination LOB.
CREATETEMPORARY Procedure on page 23-23	Creates a temporary BLOB or CLOB and its corresponding index in the user's default temporary tablespace.
ERASE Procedure on page 23-24	Erases all or part of a LOB.
FILECLOSE Procedure on page 23-26	Closes the file.
FILECLOSEALL Procedure on page 23-28	Closes all previously opened files.
FILEEXISTS Function on page 23-28	Checks if the file exists on the server.
FILEGETNAME Procedure on page 23-30	Gets the directory alias and file name.
FILEISOPEN Function on page 23-31	Checks if the file was opened using the input BFILE locators.
FILEOPEN Procedure on page 23-32	Opens a file.
FREETEMPORARY Procedure on page 23-34	Frees the temporary BLOB or CLOB in the user's default temporary tablespace.
GETCHUNKSIZE Function on page 23-35	Returns the amount of space used in the LOB chunk to store the LOB value.
GETLENGTH Function on page 23-36	Gets the length of the LOB value.
INSTR Function on page 23-37	Returns the matching position of the *nth* occurrence of the pattern in the LOB.
ISOPEN Function on page 23-40	Checks to see if the LOB was already opened using the input locator.
ISTEMPORARY Function on page 23-41	Checks if the locator is pointing to a temporary LOB.
LOADFROMFILE Procedure on page 23-42	Loads BFILE data into an internal LOB.
LOADBLOBFROMFILE Procedure on page 23-44	Loads BFILE data into an internal BLOB.

Table 23–4 DBMS_LOB Subprograms (Cont.)

Subprogram	Description
LOADCLOBFROMFILE Procedure on page 23-47	Loads BFILE data into an internal CLOB.
OPEN Procedure on page 23-50	Opens a LOB (internal, external, or temporary) in the indicated mode.
READ Procedure on page 23-51	Reads data from the LOB starting at the specified offset.
SUBSTR Function on page 23-55	Returns part of the LOB value starting at the specified offset.
TRIM Procedure on page 23-58	Trims the LOB value to the specified shorter length.
WRITE Procedure on page 23-60	Writes data to the LOB from a specified offset.
WRITEAPPEND Procedure on page 23-62	Writes a buffer to the end of a LOB.

APPEND Procedure

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

There are two overloaded APPEND procedures.

Syntax

```
DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY BLOB,
   src_lob  IN             BLOB);

DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY CLOB  CHARACTER SET ANY_CS,
   src_lob  IN             CLOB  CHARACTER SET dest_lob%CHARSET);
```

Parameters

Table 23–5 APPEND Procedure Parameters

Parameter	Description
dest_lob	Locator for the internal LOB to which the data is to be appended.

Table 23–5 APPEND Procedure Parameters

Parameter	Description
src_lob	Locator for the internal LOB from which the data is to be read.

Exceptions

Table 23–6 APPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Either the source or the destination LOB is NULL.

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open / Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Examples

```
CREATE OR REPLACE PROCEDURE Example_1a IS
    dest_lob BLOB;
    src_lob  BLOB;
BEGIN
    -- get the LOB locators
    -- note that the FOR UPDATE clause locks the row
    SELECT b_lob INTO dest_lob
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_lob INTO src_lob
        FROM lob_table
        WHERE key_value = 21;
    DBMS_LOB.APPEND(dest_lob, src_lob);
    COMMIT;
EXCEPTION
```

```
    WHEN some_exception
    THEN handle_exception;
END;

CREATE OR REPLACE PROCEDURE Example_1b IS
    dest_lob, src_lob  BLOB;
BEGIN
    -- get the LOB locators
    -- note that the FOR UPDATE clause locks the row
    SELECT b_lob INTO dest_lob
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_lob INTO src_lob
        FROM lob_table
        WHERE key_value = 12;
    DBMS_LOB.APPEND(dest_lob, src_lob);
    COMMIT;
EXCEPTION
    WHEN some_exception
    THEN handle_exception;
END;
```

CLOSE Procedure

This procedure closes a previously opened internal or external LOB.

Syntax

```
DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY BLOB);

DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);

DBMS_LOB.CLOSE (
   file_loc   IN OUT NOCOPY BFILE);
```

Errors

No error is returned if the BFILE exists but is not opened. An error is returned if the LOB is not open.

Usage Notes

CLOSE requires a round-trip to the server for both internal and external LOBs. For internal LOBs, CLOSE triggers other code that relies on the close call, and for external LOBs (BFILEs), CLOSE actually closes the server-side operating system file.

It is not mandatory that you wrap all LOB operations inside the Open / Close APIs. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

COMPARE Function

This function compares two entire LOBs or parts of two LOBs. You can only compare LOBs of the same datatype (LOBs of BLOB type with other BLOBs, and CLOBs with CLOBs, and BFILEs with BFILEs). For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

COMPARE returns zero if the data exactly matches over the range specified by the offset and amount parameters. Otherwise, a nonzero INTEGER is returned.

For fixed-width *n*-byte CLOBs, if the input amount for COMPARE is specified to be greater than (4294967295 / *n*), then COMPARE matches characters in a range of size (4294967295 / *n*), or Max(length(clob1), length(clob2)), whichever is lesser.

Syntax

```
DBMS_LOB.COMPARE (
   lob_1            IN BLOB,
   lob_2            IN BLOB,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
```

```
   lob_1            IN CLOB  CHARACTER SET ANY_CS,
   lob_2            IN CLOB  CHARACTER SET lob_1%CHARSET,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN BFILE,
   lob_2            IN BFILE,
   amount           IN INTEGER,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 23–7 COMPARE Function Parameters

Parameter	Description
lob_1	LOB locator of first target for comparison.
lob_2	LOB locator of second target for comparison.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to compare.
offset_1	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.
offset_2	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.

Returns

- INTEGER: Zero if the comparison succeeds, nonzero if not.
- NULL, if
  - amount < 1
  - amount > LOBMAXSIZE
  - offset_1 or offset_2 < 1

* offset_1 or offset_2 > LOBMAXSIZE

Exceptions

Table 23–8 COMPARE Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Examples

```
CREATE OR REPLACE PROCEDURE Example2a IS
    lob_1, lob_2      BLOB;
    retval            INTEGER;
BEGIN
    SELECT b_col INTO lob_1 FROM lob_table
        WHERE key_value = 45;
    SELECT b_col INTO lob_2 FROM lob_table
        WHERE key_value = 54;
    retval := dbms_lob.compare(lob_1, lob_2, 5600, 33482,
         128);
    IF retval = 0 THEN
      ;    -- process compared code
    ELSE
      ;    -- process not compared code
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_2b IS
    fil_1, fil_2       BFILE;
    retval             INTEGER;
BEGIN

    SELECT f_lob INTO fil_1 FROM lob_table WHERE key_value = 45;
    SELECT f_lob INTO fil_2 FROM lob_table WHERE key_value = 54;
    dbms_lob.fileopen(fil_1, dbms_lob.file_readonly);
    dbms_lob.fileopen(fil_2, dbms_lob.file_readonly);
    retval := dbms_lob.compare(fil_1, fil_2, 5600,
```

```
                                    3348276, 2765612);
    IF (retval = 0)
    THEN
        ; -- process compared code
    ELSE
        ; -- process not compared code
    END IF;
    dbms_lob.fileclose(fil_1);
    dbms_lob.fileclose(fil_2);
END;
```

COPY Procedure

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the `src_offset` to the end of the source LOB.

Syntax

```
DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY BLOB,
  src_lob     IN            BLOB,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);

DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY CLOB  CHARACTER SET ANY_CS,
  src_lob     IN            CLOB  CHARACTER SET dest_lob%CHARSET,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);
```

Parameters

Table 23–9 COPY Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the copy target.
src_lob	LOB locator of source for the copy.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to copy.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.
src_offset	Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.

Exceptions

Table 23–10 COPY Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or invalid.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1 - src_offset or dest_offset > LOBMAXSIZE - amount < 1 - amount > LOBMAXSIZE

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open / Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Examples

```
CREATE OR REPLACE PROCEDURE Example_3a IS
    lobd, lobs     BLOB;
    dest_offset    INTEGER := 1
    src_offset     INTEGER := 1
    amt            INTEGER := 3000;
BEGIN
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_col INTO lobs
        FROM lob_table
        WHERE key_value = 21;
    DBMS_LOB.COPY(lobd, lobs, amt, dest_offset, src_offset);
    COMMIT;
   EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;

CREATE OR REPLACE PROCEDURE Example_3b IS
    lobd, lobs     BLOB;
    dest_offset    INTEGER := 1
    src_offset     INTEGER := 1
    amt            INTEGER := 3000;
BEGIN
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_col INTO lobs
        FROM lob_table
        WHERE key_value = 12;
    DBMS_LOB.COPY(lobd, lobs, amt, dest_offset, src_offset);
    COMMIT;
   EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

CREATETEMPORARY Procedure

This procedure creates a temporary BLOB or CLOB and its corresponding index in your default temporary tablespace.

Syntax

```
DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY BLOB,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);

DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);
```

Parameters

Table 23–11 CREATETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
cache	Specifies if LOB should be read into buffer cache or not.
dur	1 of 2 predefined duration values (SESSION or CALL) which specifies a hint as to whether the temporary LOB is cleaned up at the end of the session or call. If dur is omitted, then the session duration is used.

Example

```
DBMS_LOB.CREATETEMPORARY(Dest_Loc, TRUE)
```

See Also: *PL/SQL User's Guide and Reference* for more information about NOCOPY and passing temporary lobs as parameters.

ERASE Procedure

This procedure erases an entire internal LOB or part of an internal LOB.

Note: The length of the LOB is not decreased when a section of the LOB is erased. To decrease the length of the LOB value, see the "TRIM Procedure" on page 23-58.

When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for BLOBs or CLOBs respectively.

The actual number of bytes or characters erased can differ from the number you specified in the amount parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the amount parameter.

Syntax

```
DBMS_LOB.ERASE (
   lob_loc          IN OUT   NOCOPY   BLOB,
   amount           IN OUT   NOCOPY   INTEGER,
   offset           IN                INTEGER := 1);

DBMS_LOB.ERASE (
   lob_loc          IN OUT   NOCOPY   CLOB CHARACTER SET ANY_CS,
   amount           IN OUT   NOCOPY   INTEGER,
   offset           IN                INTEGER := 1);
```

Parameters

Table 23–12 ERASE Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be erased.
amount	Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.
offset	Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).

Exceptions

Table 23–13 ERASE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any input parameter is NULL.
INVALID_ARGVAL	Either: - amount < 1 or amount > LOBMAXSIZE - offset < 1 or offset > LOBMAXSIZE

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open / Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

Example

```
CREATE OR REPLACE PROCEDURE Example_4 IS
    lobd        BLOB;
    amt         INTEGER := 3000;
BEGIN
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    dbms_lob.erase(dest_lob, amt, 2000);
    COMMIT;
END;
```

See Also: "TRIM Procedure" on page 23-58

FILECLOSE Procedure

This procedure closes a `BFILE` that has already been opened through the input locator.

> **Note:** Oracle has only read-only access to `BFILEs`. This means that `BFILEs` cannot be written through Oracle.

Syntax

```
DBMS_LOB.FILECLOSE (
    file_loc IN OUT NOCOPY BFILE);
```

Parameters

Table 23–14 FILECLOSE Procedure Parameter

Parameter	Description
file_loc	Locator for the BFILE to be closed.

Exceptions

Table 23–15 FILECLOSE Procedure Exceptions

Exception	Description
VALUE_ERROR	NULL input value for file_loc.
UNOPENED_FILE	File was not opened with the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Example

```
CREATE OR REPLACE PROCEDURE Example_5 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil);
    -- file operations
    dbms_lob.fileclose(fil);
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also:

- "FILEOPEN Procedure" on page 23-32
- "FILECLOSEALL Procedure" on page 23-28

FILECLOSEALL Procedure

This procedure closes all BFILEs opened in the session.

Syntax

```
DBMS_LOB.FILECLOSEALL;
```

Exceptions

Table 23–16 FILECLOSEALL Procedure Exception

Exception	Description
UNOPENED_FILE	No file has been opened in the session.

Example

```
CREATE OR REPLACE PROCEDURE Example_6 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil);
    -- file operations
    dbms_lob.filecloseall;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also:

- "FILEOPEN Procedure" on page 23-32
- "FILECLOSE Procedure" on page 23-26

FILEEXISTS Function

This function finds out if a given BFILE locator points to a file that actually exists on the server's file system.

Syntax

```
DBMS_LOB.FILEEXISTS (
   file_loc     IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 23–17 FILEEXISTS Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Returns

Table 23–18 FILEEXISTS Function Returns

Return	Description
0	Physical file does not exist.
1	Physical file exists.

Exceptions

Table 23–19 FILEEXISTS Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

Example

```
CREATE OR REPLACE PROCEDURE Example_7 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    IF (dbms_lob.fileexists(fil))
    THEN
        ; -- file exists code
    ELSE
        ; -- file does not exist code
    END IF;
    EXCEPTION
```

```
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "FILEISOPEN Function" on page 23-31.

FILEGETNAME Procedure

This procedure determines the directory alias and filename, given a BFILE locator. This function only indicates the directory alias name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the dir_alias buffer is 30, and for the entire path name, it is 2000.

Syntax

```
DBMS_LOB.FILEGETNAME (
   file_loc   IN    BFILE,
   dir_alias  OUT   VARCHAR2,
   filename   OUT   VARCHAR2);
```

Parameters

Table 23–20 FILEGETNAME Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE.
dir_alias	Directory alias.
filename	Name of the BFILE.

Exceptions

Table 23–21 FILEGETNAME Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	dir_alias or filename are NULL.

Example

```
CREATE OR REPLACE PROCEDURE Example_8 IS
```

```
    fil BFILE;
    dir_alias VARCHAR2(30);
    name VARCHAR2(2000);
BEGIN
    IF (dbms_lob.fileexists(fil))
    THEN
        dbms_lob.filegetname(fil, dir_alias, name);
        dbms_output.put_line("Opening " || dir_alias || name);
        dbms_lob.fileopen(fil, dbms_lob.file_readonly);
        -- file operations
        dbms_output.fileclose(fil);
    END IF;
END;
```

FILEISOPEN Function

This function finds out whether a BFILE was opened with the given FILE locator.

If the input FILE locator was never passed to the FILEOPEN procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

Syntax

```
DBMS_LOB.FILEISOPEN (
   file_loc   IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEISOPEN, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 23–22 FILEISOPEN Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Returns

INTEGER: 0 = file is not open, 1 = file is open

Exceptions

Table 23–23 FILEISOPEN Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

Example

```
CREATE OR REPLACE PROCEDURE Example_9 IS
DECLARE
    fil      BFILE;
    pos      INTEGER;
    pattern  VARCHAR2(20);
BEGIN
    SELECT f_lob INTO fil FROM lob_table
        WHERE key_value = 12;
    -- open the file
    IF (dbms_lob.fileisopen(fil))
    THEN
        pos := dbms_lob.instr(fil, pattern, 1025, 6);
         -- more file operations
         dbms_lob.fileclose(fil);
   ELSE
        ; -- return error
    END IF;
END;
```

See Also: "FILEEXISTS Function" on page 23-28

FILEOPEN Procedure

This procedure opens a BFILE for read-only access. BFILEs may not be written through Oracle.

Syntax

```
DBMS_LOB.FILEOPEN (
   file_loc   IN OUT NOCOPY  BFILE,
   open_mode  IN             BINARY_INTEGER := file_readonly);
```

Parameters

Table 23–24 FILEOPEN Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE.
open_mode	File access is read-only.

Exceptions

Table 23–25 FILEOPEN Procedure Exceptions

Exception	Description
VALUE_ERROR	file_loc or open_mode is NULL.
INVALID_ARGVAL	open_mode is not equal to FILE_READONLY.
OPEN_TOOMANY	Number of open files in the session exceeds session_max_open_files.
NOEXIST_DIRECTORY	Directory associated with file_loc does not exist.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Example

```
CREATE OR REPLACE PROCEDURE Example_10 IS
    fil BFILE;
BEGIN
    -- open BFILE
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    IF (dbms_lob.fileexists(fil))
    THEN
        dbms_lob.fileopen(fil, dbms_lob.file_readonly);
        -- file operation
        dbms_lob.fileclose(fil);
    END IF;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also:

- "FILECLOSE Procedure" on page 23-26
- "FILECLOSEALL Procedure" on page 23-28

FREETEMPORARY Procedure

This procedure frees the temporary BLOB or CLOB in your default temporary tablespace. After the call to FREETEMPORARY, the LOB locator that was freed is marked as invalid.

If an invalid LOB locator is assigned to another LOB locator using OCILobLocatorAssign in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

Syntax

```
DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 23–26 FREETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.

Example

```
DECLARE
  a blob;
  b blob;
BEGIN
  dbms_lob.createtemporary(a, TRUE);
  dbms_lob.createtemporary(b, TRUE);
  ...
  -- the following call frees lob a
  dbms_lob.freetemporary(a);
  -- at this point lob locator a is marked as invalid
  -- the following assignment frees the lob b and marks it as invalid
also
```

```
  b := a;
END;
```

GETCHUNKSIZE Function

When creating the table, you can specify the chunking factor, which can be a multiple of Oracle blocks. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value.

This function returns the amount of space used in the LOB chunk to store the LOB value.

Syntax

```
DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETCHUNKSIZE, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 23–27 GETCHUNKSIZE Function Parameters

Parameter	Description
lob_loc	LOB locator.

Returns

The value returned for BLOBs is in terms of bytes. The value returned for CLOBs is in terms of characters.

Usage Notes

Performance is improved if you enter read / write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is

done or duplicated. You could batch up the WRITE until you have enough for a chunk, instead of issuing several WRITE calls for the same chunk.

GETLENGTH Function

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a BFILE includes the EOF, if it exists. Any 0-byte or space filler in the LOB caused by previous ERASE or WRITE operations is also included in the length count. The length of an empty internal LOB is 0.

Syntax

```
DBMS_LOB.GETLENGTH (
   lob_loc    IN  BLOB)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   lob_loc    IN  CLOB   CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   file_loc    IN  BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 23–28 GETLENGTH Function Parameter

Parameter	Description
file_loc	The file locator for the LOB whose length is to be returned.

Returns

The length of the LOB in bytes or characters as an INTEGER. NULL is returned if the input LOB is NULL or if the input lob_loc is NULL. An error is returned in the following cases for BFILEs:

- lob_loc does not have the necessary directory and operating system privileges
- lob_loc cannot be read because of an operating system read error

Examples

```
CREATE OR REPLACE PROCEDURE Example_11a IS
    lobd        BLOB;
    length      INTEGER;
BEGIN
    -- get the LOB locator
    SELECT b_lob INTO lobd FROM lob_table
        WHERE key_value = 42;
    length := dbms_lob.getlength(lobd);
    IF length IS NULL THEN
        dbms_output.put_line('LOB is null.');
    ELSE
        dbms_output.put_line('The length is '
            || length);
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_11b IS
DECLARE
    len INTEGER;
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    len := dbms_lob.length(fil);
END;
```

INSTR Function

This function returns the matching position of the *nth* occurrence of the pattern in the LOB, starting from the offset you specify.

The form of the VARCHAR2 buffer (the pattern parameter) must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

Operations that accept RAW or VARCHAR2 parameters for pattern matching, such as INSTR, do not support regular expressions or special matching characters (as in the case of SQL LIKE) in the pattern parameter or substrings.

Syntax

```
DBMS_LOB.INSTR (
   lob_loc    IN   BLOB,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   lob_loc    IN   CLOB      CHARACTER SET ANY_CS,
   pattern    IN   VARCHAR2  CHARACTER SET lob_loc%CHARSET,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   file_loc   IN   BFILE,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 23–29 INSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be examined.
file_loc	The file locator for the LOB to be examined.
pattern	Pattern to be tested for. The pattern is a group of RAW bytes for BLOBs, and a character string (VARCHAR2) for CLOBs.The maximum size of the pattern is 16383 bytes.
offset	Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)

Table 23–29 INSTR Function Parameters

Parameter	Description
nth	Occurrence number, starting at 1.

Returns

Table 23–30 INSTR Function Returns

Return	Description
INTEGER	Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.
NULL	Either: -any one or more of the IN parameters was NULL or INVALID. -offset < 1 or offset > LOBMAXSIZE. -nth < 1. -nth > LOBMAXSIZE.

Exceptions

Table 23–31 INSTR Function Exceptions for BFILES

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Examples

```
CREATE OR REPLACE PROCEDURE Example_12a IS
    lobd        CLOB;
    pattern     VARCHAR2 := 'abcde';
    position    INTEGER  := 10000;
BEGIN
-- get the LOB locator
    SELECT b_col INTO lobd
```

```
        FROM lob_table
        WHERE key_value = 21;
    position := DBMS_LOB.INSTR(lobd,
                        pattern, 1025, 6);
    IF position = 0 THEN
        dbms_output.put_line('Pattern not found');
    ELSE
        dbms_output.put_line('The pattern occurs at '
                || position);
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_12b IS
DECLARE
    fil BFILE;
    pattern VARCHAR2;
    pos INTEGER;
BEGIN
    -- initialize pattern
    -- check for the 6th occurrence starting from 1025th byte
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    pos := dbms_lob.instr(fil, pattern, 1025, 6);
    dbms_lob.fileclose(fil);
END;
```

See Also: "SUBSTR Function" on page 23-55

ISOPEN Function

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

Syntax

```
DBMS_LOB.ISOPEN (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   file_loc IN BFILE)
```

```
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(ISOPEN, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 23–32 ISOPEN Function Parameters

Parameter	Description
lob_loc	LOB locator.
file_loc	File locator.

Usage Notes

For BFILES, openness is associated with the locator. If the input locator was never passed to OPEN, the BFILE is not considered to be opened by this locator. However, a different locator may have opened the BFILE. More than one OPEN can be performed on the same BFILE using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, ISOPEN requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (BFILEs), ISOPEN also requires a round-trip, because that's where the state is kept.

ISTEMPORARY Function

Syntax

```
DBMS_LOB.ISTEMPORARY (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISTEMPORARY (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 23–33 ISTEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
temporary	Boolean, which indicates whether the LOB is temporary or not.

Returns

This function returns TRUE in temporary if the locator is pointing to a temporary LOB. It returns FALSE otherwise.

LOADFROMFILE Procedure

This procedure copies all, or a part of, a source external LOB (BFILE) to a destination internal LOB.

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is either in bytes or characters for BLOBs and CLOBs respectively.

> **Note:** The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE.

Note: If the character set is varying width, UTF-8 for example, the LOB value is stored in the fixed-width UCS2 format. Therefore, if you are using DBMS_LOB.LOADFROMFILE, the data in the BFILE should be in the UCS2 character set instead of the UTF-8 character set. However, you should use sql*loader instead of LOADFROMFILE to load data into a CLOB or NCLOB because sql*loader will provide the necessary character set conversions.

Syntax

```
DBMS_LOB.LOADFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_file    IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN            INTEGER  := 1,
   src_offset  IN            INTEGER  := 1);
```

Parameters

Table 23–34 LOADFROMFILE Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the target for the load.
src_file	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.
src_offset	Offset in bytes in the source BFILE (origin: 1) for the start of the load.

Usage Requirements

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Exceptions

Table 23–35 LOADFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

Example

```
CREATE OR REPLACE PROCEDURE Example_l2f IS
  lobd       BLOB;
  fils       BFILE   := BFILENAME('SOME_DIR_OBJ','some_file');
  amt        INTEGER := 4000;
BEGIN
  SELECT b_lob INTO lobd FROM lob_table WHERE key_value = 42 FOR UPDATE;
  dbms_lob.fileopen(fils, dbms_lob.file_readonly);
  dbms_lob.loadfromfile(lobd, fils, amt);
  COMMIT;
  dbms_lob.fileclose(fils);
```

LOADBLOBFROMFILE Procedure

This procedure loads data from BFILE to internal BLOB. This achieves the same outcome as LOADFROMFILE, and returns the new offsets.

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in bytes for BLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination

BLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).

Syntax

```
DBMS_LOB.LOADBLOBFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_bfile   IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN OUT        INTEGER,
   src_offset  IN OUT        INTEGER);
```

Parameters

Table 23–36 LOADBLOBFROMFILE Procedure Parameters

Parameter	Description
dest_lob	BLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. You can also use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.
dest_offset	(IN) Offset in bytes in the destination BLOB (origin: 1) for the start of the write. (OUT) New offset in bytes in the destination BLOB right after the end of this write, which is also where the next write should begin.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read.(OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.

Usage Requirements

It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for IN/OUT parameter or provide a default value for the IN parameter. Here is a summary of the constants and the defaults that can be used.

Table 23–37 Suggested Values of the Parameter

Parameter	Default Value	Description
amount	DBMSLOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning

Constants defined in DBMSLOB.SQL

```
lobmaxsize                      CONSTANT INTEGER        := 4294967295;
```

Exceptions

Table 23–38 LOADBLOBFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

Example

```
TBD
;
```

LOADCLOBFROMFILE Procedure

This procedure loads data from a BFILE to an internal CLOB/NCLOB with necessary character set conversion and returns the new offsets.

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in characters for CLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination CLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).

Syntax

```
DBMS_LOB.LOADCLOBFROMFILE (
   dest_lob       IN OUT NOCOPY   BLOB,
   src_bfile      IN              BFILE,
   amount         IN              INTEGER,
   dest_offset    IN OUT          INTEGER,
   src_offset     IN OUT          INTEGER,
   src_csid       IN              NUMBER,
   lang_context   IN OUT          INTEGER,
   warning        OUT             INTEGER);
```

Parameters

Table 23–39 LOADCLOBFROMFILE Procedure Parameters

Parameter	Description
dest_lob	CLOB/NCLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. Use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.

Table 23–39 LOADCLOBFROMFILE Procedure Parameters

Parameter	Description
dest_offset	(IN) Offset in characters in the destination CLOB (origin: 1) for the start of the write. (OUT) The new offset in characters right after the end of this load, which is also where the next load should start. It always points to the beginning of the first complete character after the end of load. If the last character is not complete, offset goes back to the beginning of the partial character.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.
src_csid	Character set id of the source (BFILE) file.
lang_context	(IN) Language context, such as shift status, for the current load. (OUT) The language context at the time when the current load stopped, and what the next load should be using if continuing loading from the same source. This information is returned to the user so that they can use it for the continuous load without losing or misinterpreting any source data. For the very first load or if do not care, simply use the default 0. The details of this language context is hidden from the user. One does not need to know what it is or what's in it in order to make the call
warning	(OUT) Warning message. This indicates something abnormal happened during the loading. It may or may not be caused by the user's mistake. The loading is completed as required, and it's up to the user to check the warning message. Currently, the only possible warning is the inconvertible character. This happens when the character in the source cannot be properly converted to a character in destination, and the default replacement character (e.g., '?') is used in place. The message is defined as warn_inconvertable_char in DBMSLOB.

Usage Requirements

- The destination character set is always the same as the database character set in the case of CLOB and national character set in the case of NCLOB.
- csid=0 indicates the default behavior that uses database csid for CLOB and national csid for NCLOB in the place of source csid. Conversion is still necessary if it is of varying width

- It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

  If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for IN/OUT parameter or give a default value for the IN parameter. Here is a summary of the constants and the defaults that can be used.

Table 23–40 Suggested Values of the Parameter

Parameter	Default Value	Description
amount	DBMSLOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
csid	0 (IN)	default csid, use destination csid
lang_context	0 (IN)	default language context
warning	0 (OUT)	no warning message, everything is ok

Constants defined in DBMSLOB.SQL

```
lobmaxsize                  CONSTANT INTEGER      := 4294967295;
warn_inconvertible_char     CONSTANT INTEGER      := 1;
default_csid                CONSTANT INTEGER      := 0;
default_lang_ctx            CONSTANT INTEGER      := 0;
no_warning                  CONSTANT INTEGER      := 0;
```

Exceptions

Table 23–41 LOADCLOBFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

Example

```
TBD
;
```

OPEN Procedure

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read / write. It is an error to open the same LOB twice.

> **Note:** If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. BFILE can only be opened with read-only mode.

In Oracle8.0, the constant file_readonly was the only valid mode in which to open a BFILE. For Oracle 8*i*, two new constants have been added to the DBMS_LOB package: lob_readonly and lob_readwrite.

Syntax

```
DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY BLOB,
   open_mode IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   open_mode IN            BINARY_INTEGER);
```

```
DBMS_LOB.OPEN (
   file_loc  IN OUT NOCOPY BFILE,
   open_mode IN            BINARY_INTEGER := file_readonly);
```

Parameters

Table 23–42 OPEN Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
open_mode	Mode in which to open.

Usage Notes

OPEN requires a roundtrip to the server for both internal and external LOBs. For internal LOBs, OPEN triggers other code that relies on the OPEN call. For external LOBs (BFILEs), OPEN requires a round-trip because the actual operating system file on the server side is being opened.

It is not mandatory that you wrap all LOB operations inside the Open/Close APIs. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and nonLOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

READ Procedure

This procedure reads a piece of a LOB, and returns the specified amount into the buffer parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the amount parameter. If the input offset points past the End of LOB, then amount is set to 0, and a NO_DATA_FOUND exception is raised.

Syntax

```
DBMS_LOB.READ (
```

```
   lob_loc   IN             BLOB,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            RAW);

DBMS_LOB.READ (
   lob_loc   IN             CLOB CHARACTER SET ANY_CS,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
   file_loc   IN              BFILE,
   amount    IN OUT   NOCOPY BINARY_INTEGER,
   offset    IN               INTEGER,
   buffer    OUT              RAW);
```

Parameters

Table 23–43 READ Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read.
file_loc	The file locator for the LOB to be examined.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).
buffer	Output buffer for the read operation.

Exceptions

Table 23–44 READ Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL.

Table 23–44 READ Procedure Exceptions

Exception	Description
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE - amount is greater, in bytes or characters, than the capacity of buffer.
NO_DATA_FOUND	End of the LOB is reached, and there are no more bytes or characters to read from the LOB: amount has a value of 0.

Exceptions

Table 23–45 READ Procedure Exceptions for BFILEs

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.READ from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. Oracle converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

Examples

```
CREATE OR REPLACE PROCEDURE Example_13a IS
```

```
    src_lob        BLOB;
    buffer         RAW(32767);
    amt            BINARY_INTEGER := 32767;
    pos            INTEGER := 2147483647;
BEGIN
    SELECT b_col INTO src_lob
        FROM lob_table
        WHERE key_value = 21;
    LOOP
        dbms_lob.read (src_lob, amt, pos, buffer);
        -- process the buffer
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            dbms_output.put_line('End of data');
END;

CREATE OR REPLACE PROCEDURE Example_13b IS
    fil BFILE;
    buf RAW(32767);
    amt BINARY_INTEGER := 32767;
    pos INTEGER := 2147483647;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    LOOP
        dbms_lob.read(fil, amt, pos, buf);
        -- process contents of buf
        pos := pos + amt;
    END LOOP;
    EXCEPTION
      WHEN NO_DATA_FOUND
      THEN
      BEGIN
        dbms_output.putline ('End of LOB value reached');
        dbms_lob.fileclose(fil);
      END;
END;
```

Example for efficient operating system I/O that performs better with block I/O rather than stream I/O:

```
CREATE OR REPLACE PROCEDURE Example_13c IS
    fil BFILE;
```

```
    amt BINARY_INTEGER := 1024; -- or n x 1024 for reading n
    buf RAW(1024); -- blocks at a time
    tmpamt BINARY_INTEGER;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    LOOP
        dbms_lob.read(fil, amt, pos, buf);
        -- process contents of buf
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
            BEGIN
                dbms_output.putline ('End of data reached');
                dbms_lob.fileclose(fil);
            END;
END;
```

SUBSTR Function

This function returns amount bytes or characters of a LOB, starting from an absolute offset from the beginning of the LOB.

For fixed-width n-byte CLOBs, if the input amount for SUBSTR is specified to be greater than (32767 / n), then SUBSTR returns a character buffer of length (32767 / n), or the length of the CLOB, whichever is lesser. For CLOBs in a varying-width character set, n is 2.

Syntax

```
DBMS_LOB.SUBSTR (
   lob_loc     IN    BLOB,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;

DBMS_LOB.SUBSTR (
   lob_loc     IN    CLOB   CHARACTER SET ANY_CS,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;

DBMS_LOB.SUBSTR (
```

```
  file_loc     IN    BFILE,
  amount       IN    INTEGER := 32767,
  offset       IN    INTEGER := 1)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 23–46 SUBSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read.
file_loc	The file locator for the LOB to be examined.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to be read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).

Returns

Table 23–47 SUBSTR Function Returns

Return	Description
RAW	Function overloading that has a BLOB or BFILE in parameter.
VARCHAR2	CLOB version.
NULL	Either: - any input parameter is NULL - amount < 1 - amount > 32767 - offset < 1 - offset > LOBMAXSIZE

Exceptions

Table 23–48 SUBSTR Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. Oracle converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

Examples

```
CREATE OR REPLACE PROCEDURE Example_14a IS
    src_lob        CLOB;
    pos            INTEGER := 2147483647;
    buf            VARCHAR2(32000);
BEGIN
    SELECT c_lob INTO src_lob FROM lob_table
        WHERE key_value = 21;
    buf := DBMS_LOB.SUBSTR(src_lob, 32767, pos);
    -- process the data
END;

CREATE OR REPLACE PROCEDURE Example_14b IS
    fil BFILE;
    pos INTEGER := 2147483647;
    pattern RAW;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
```

```
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    pattern := dbms_lob.substr(fil, 255, pos);
    dbms_lob.fileclose(fil);
END;
```

See Also:

- "INSTR Function" on page 23-37
- "READ Procedure" on page 23-51

TRIM Procedure

This procedure trims the value of the internal LOB to the length you specify in the newlen parameter. Specify the length in bytes for BLOBs, and specify the length in characters for CLOBs.

Note: The TRIM procedure decreases the length of the LOB to the value specified in the newlen parameter.

If you attempt to TRIM an empty LOB, then nothing occurs, and TRIM returns no error. If the new length that you specify in newlen is greater than the size of the LOB, then an exception is raised.

Syntax

```
DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY BLOB,
   newlen          IN              INTEGER);

DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS,
   newlen          IN              INTEGER);
```

Parameters

Table 23–49 TRIM Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB whose length is to be trimmed.
newlen	New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.

Exceptions

Table 23–50 TRIM Procedure Exceptions

Exception	Description
VALUE_ERROR	lob_loc is NULL.
INVALID_ARGVAL	Either: - new_len < 0 - new_len > LOBMAXSIZE

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open / Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Example

```
CREATE OR REPLACE PROCEDURE Example_15 IS
    lob_loc        BLOB;
BEGIN
-- get the LOB locator
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 42 FOR UPDATE;
    dbms_lob.trim(lob_loc, 4000);
    COMMIT;
END;
```

See Also:

- "ERASE Procedure" on page 23-24
- "WRITEAPPEND Procedure" on page 23-62

WRITE Procedure

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the buffer parameter.

WRITE replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.

Syntax

```
DBMS_LOB.WRITE (
   lob_loc  IN OUT NOCOPY  BLOB,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             RAW);

DBMS_LOB.WRITE (
   lob_loc  IN OUT  NOCOPY CLOB   CHARACTER SET ANY_CS,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 23–51 WRITE Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.
buffer	Input buffer for the write.

Exceptions

Table 23–52 WRITE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITE from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. Oracle converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Example

```
CREATE OR REPLACE PROCEDURE Example_16 IS
    lob_loc         BLOB;
    buffer          RAW;
```

```
    amt             BINARY_INTEGER := 32767;
    pos             INTEGER := 2147483647;
    i               INTEGER;
BEGIN
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    FOR i IN 1..3 LOOP
        dbms_lob.write (lob_loc, amt, pos, buffer);
        -- fill in more data
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also:

- "APPEND Procedure" on page 23-15
- "COPY Procedure" on page 23-21

WRITEAPPEND Procedure

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the `buffer` parameter.

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

Syntax

```
DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY BLOB,
   amount  IN            BINARY_INTEGER,
   buffer  IN            RAW);

DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount  IN            BINARY_INTEGER,
   buffer  IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 23–53 WRITEAPPEND Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
buffer	Input buffer for the write.

Exceptions

Table 23–54 WRITEAPPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITEAPPEND from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. Oracle converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

It is not mandatory that you wrap the LOB operation inside the Open/Close APIs. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open / Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

Example

```
CREATE OR REPLACE PROCEDURE Example_17 IS
    lob_loc    BLOB;
    buffer     RAW;
    amt        BINARY_INTEGER := 32767;
    i          INTEGER;
BEGIN
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    FOR i IN 1..3 LOOP
        -- fill the buffer with data to be written to the lob
        dbms_lob.writeappend (lob_loc, amt, buffer);
    END LOOP;
END;
```

See Also:

- "APPEND Procedure" on page 23-15
- "COPY Procedure" on page 23-21
- "WRITE Procedure" on page 23-60

24

DBMS_LOCK

Oracle Lock Management services for your applications are available through procedures in the DBMS_LOCK package. You can request a lock of a specific mode, give it a unique name recognizable in another procedure in the same or another instance, change the lock mode, and release it.

Because a reserved user lock is the same as an Oracle lock, it has all the functionality of an Oracle lock, such as deadlock detection. Be certain that any user locks used in distributed transactions are released upon COMMIT, or an undetected deadlock may occur.

User locks never conflict with Oracle locks because they are identified with the prefix "UL". You can view these locks using the Enterprise Manager lock monitor screen or the appropriate fixed views. User locks are automatically released when a session terminates.

The lock identifier is a number in the range of 0 to 1073741823.

Some uses of user locks:

- Providing exclusive access to a device, such as a terminal
- Providing application-level enforcement of read locks
- Detecting when a lock is released and cleanup after the application
- Synchronizing applications and enforcing sequential processing

This chapter discusses the following topics:

Requirements, Security, and Constants for DBMS_LOCK

Requirements

DBMS_LOCK is most efficient with a limit of a few hundred locks for each session. Oracle strongly recommends that you develop a standard convention for using these locks in order to avoid conflicts among procedures trying to use the same locks. For example, include your company name as part of your lock names.

Security

There might be operating system-specific limits on the maximum number of total locks available. This *must* be considered when using locks or making this package available to other users. Consider granting the EXECUTE privilege only to specific users or roles.

A better alternative would be to create a cover package limiting the number of locks used and grant EXECUTE privilege to specific users. An example of a cover package is documented in the DBMSLOCK.SQL package specification file.

Constants

```
nl_mode  constant integer := 1;
ss_mode  constant integer := 2;       -- Also called 'Intended Share'
sx_mode  constant integer := 3;       -- Also called 'Intended Exclusive'
s_mode   constant integer := 4;
ssx_mode constant integer := 5;
x_mode   constant integer := 6;
```

These are the various lock modes (nl -> "NuLl", ss -> "Sub Shared", sx -> "Sub eXclusive", s -> "Shared", ssx -> "Shared Sub eXclusive", x -> "eXclusive").

A sub-share lock can be used on an aggregate object to indicate that share locks are being aquired on sub-parts of the object. Similarly, a sub-exclusive lock can be used on an aggregate object to indicate that exclusive locks are being aquired on sub-parts of the object. A share-sub-exclusive lock indicates that the entire aggregate object has a share lock, but some of the sub-parts may additionally have exclusive locks.

Lock Compatibility Rules

When another process holds "held", an attempt to get "get" does the following:

Table 24–1 Lock Compatibility

HELD MODE	GET NL	GET SS	GET SX	GET S	GET SSX	GET X
NL	Success	Success	Success	Success	Success	Success
SS	Success	Success	Success	Success	Success	Fail
SX	Success	Success	Success	Fail	Fail	Fail
S	Success	Success	Fail	Success	Fail	Fail
SSX	Success	Success	Fail	Fail	Fail	Fail
X	Success	Fail	Fail	Fail	Fail	Fail

```
maxwait  constant integer := 32767;
```

The constant `maxwait` waits forever.

Summary of DBMS_LOCK Subprograms

Table 24–2 DBMS_LOCK Package Subprograms

Subprogram	Description
ALLOCATE_UNIQUE Procedure on page 24-3	Allocates a unique lock ID to a named lock.
REQUEST Function on page 24-5	Requests a lock of a specific mode.
CONVERT Function on page 24-7	Converts a lock from one mode to another.
RELEASE Function on page 24-8	Releases a lock.
SLEEP Procedure on page 24-9	Puts a procedure to sleep for a specific time.

ALLOCATE_UNIQUE Procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) given a lock name. Lock identifiers are used to enable applications to coordinate their use of locks. This is provided because it may be easier for applications to coordinate their use of locks based on lock names rather than lock numbers.

If you choose to identify locks by name, you can use ALLOCATE_UNIQUE to generate a unique lock identification number for these named locks.

The first session to call ALLOCATE_UNIQUE with a new lock name causes a unique lock ID to be generated and stored in the dbms_lock_allocated table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least expiration_secs (defaults to 10 days) past the last call to ALLOCATE_UNIQUE with the given lock name. After this time, the row in the dbms_lock_allocated table for this lock name may be deleted in order to recover space. ALLOCATE_UNIQUE performs a commit.

Caution: Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a given name.

Syntax

```
DBMS_LOCK.ALLOCATE_UNIQUE (
   lockname         IN  VARCHAR2,
   lockhandle       OUT VARCHAR2,
   expiration_secs  IN  INTEGER   DEFAULT 864000);
```

Parameters

Table 24–3 ALLOCATE_UNIQUE Procedure Parameters

Parameter	Description
lockname	Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with ORA$; these are reserved for products supplied by Oracle Corporation.

Table 24–3 ALLOCATE_UNIQUE Procedure Parameters

Parameter	Description
lockhandle	Returns the handle to the lock ID generated by ALLOCATE_UNIQUE. You can use this handle in subsequent calls to REQUEST, CONVERT, and RELEASE. A handle is returned instead of the actual lock ID to reduce the chance that a programming error accidentally creates an incorrect, but valid, lock ID. This provides better isolation between different applications that are using this package. LOCKHANDLE can be up to VARCHAR2 (128). All sessions using a lock handle returned by ALLOCATE_UNIQUE with the same lock name are referring to the same lock. Therefore, do not pass lock handles from one session to another.
expiration_specs	Number of seconds to wait after the last ALLOCATE_UNIQUE has been performed on a given lock, before permitting that lock to be deleted from the DBMS_LOCK_ALLOCATED table. The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to ALLOCATE_UNIQUE may delete expired locks to recover space.

Errors

ORA-20000, ORU-10003: Unable to find or insert lock <lockname> into catalog dbms_lock_allocated.

REQUEST Function

This function requests a lock with a given mode. REQUEST is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.REQUEST(
   id                IN  INTEGER ||
   lockhandle        IN  VARCHAR2,
   lockmode          IN  INTEGER DEFAULT X_MODE,
   timeout           IN  INTEGER DEFAULT MAXWAIT,
   release_on_commit IN  BOOLEAN DEFAULT FALSE,
  RETURN INTEGER;
```

The current default values, such as X_MODE and MAXWAIT, are defined in the DBMS_LOCK package specification.

Parameters

Table 24–4 REQUEST Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.
lockmode	Mode that you are requesting for the lock. The available modes and their associated integer identifiers follow. The abbreviations for these locks, as they appear in the V$ views and Enterprise Manager monitors are in parentheses. 1 - null mode 2 - row share mode (ULRS) 3 - row exclusive mode (ULRX) 4 - share mode (ULS) 5 - share row exclusive mode (ULRSX) 6 - exclusive mode (ULX)
timeout	Number of seconds to continue trying to grant the lock. If the lock cannot be granted within this time period, then the call returns a value of 1 (timeout).
release_on_commit	Set this parameter to TRUE to release the lock on commit or roll-back. Otherwise, the lock is held until it is explicitly released or until the end of the session.

Return Values

Table 24–5 REQUEST Function Return Values

Return Value	Description
0	Success
1	Timeout

Table 24–5 REQUEST Function Return Values

Return Value	Description
2	Deadlock
3	Parameter error
4	Already own lock specified by `id` or `lockhandle`
5	Illegal lock handle

CONVERT Function

This function converts a lock from one mode to another. `CONVERT` is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the `ALLOCATE_UNIQUE` procedure.

Syntax

```
DBMS_LOCK.CONVERT(
   id         IN INTEGER ||
   lockhandle IN VARCHAR2,
   lockmode   IN INTEGER,
   timeout    IN NUMBER DEFAULT MAXWAIT)
  RETURN INTEGER;
```

Parameters

Table 24–6 CONVERT Function Parameters

Parameter	Description
`id or lockhandle`	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by `ALLOCATE_UNIQUE`, of the lock mode you want to change.

Table 24–6 CONVERT Function Parameters

Parameter	Description
lockmode	New mode that you want to assign to the given lock. The available modes and their associated integer identifiers follow. The abbreviations for these locks, as they appear in the V$ views and Enterprise Manager monitors are in parentheses. 1 - null mode 2 - row share mode (ULRS) 3 - row exclusive mode (ULRX) 4 - share mode (ULS) 5 - share row exclusive mode (ULRSX) 6 - exclusive mode (ULX)
timeout	Number of seconds to continue trying to change the lock mode. If the lock cannot be converted within this time period, then the call returns a value of 1 (timeout).

Return Values

Table 24–7 CONVERT Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Don't own lock specified by id or lockhandle
5	Illegal lock handle

RELEASE Function

This function explicitly releases a lock previously acquired using the REQUEST function. Locks are automatically released at the end of a session. RELEASE is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.RELEASE (
   id         IN INTEGER)
  RETURN INTEGER;

DBMS_LOCK.RELEASE (
   lockhandle IN VARCHAR2)
  RETURN INTEGER;
```

Parameters

Table 24–8 RELEASE Function Parameter

Parameter	Description
`id or lockhandle`	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by `ALLOCATE_UNIQUE`, of the lock mode you want to change.

Return Values

Table 24–9 RELEASE Function Return Values

Return Value	Description
0	Success
3	Parameter error
4	Do not own lock specified by `id` or `lockhandle`
5	Illegal lock handle

SLEEP Procedure

This procedure suspends the session for a given period of time.

Syntax

```
DBMS_LOCK.SLEEP (
   seconds  IN NUMBER);
```

Parameters

Table 24–10 SLEEP Procedure Parameters

Parameter	Description
seconds	Amount of time, in seconds, to suspend the session. The smallest increment can be entered in hundredths of a second; for example, 1.95 is a legal time value.

Printing a Check: Example

The following Pro*COBOL precompiler example shows how locks are used to ensure that there are no conflicts when multiple people need to access a single device. The DBMS_LOCK package is used to ensure exclusive access.

Any cashier can issue a refund to a customer returning goods. Refunds under $50 are given in cash; anything above that is given by check. This code prints the check. One printer is opened by all the cashiers to avoid the overhead of opening and closing it for every check. Therefore, lines of output from multiple cashiers can become interleaved without exclusive access to the printer.

```
CHECK-PRINT

Get the lock "handle" for the printer lock:

   MOVE "CHECKPRINT" TO LOCKNAME-ARR.
   MOVE 10 TO LOCKNAME-LEN.
   EXEC SQL EXECUTE
      BEGIN DBMS_LOCK.ALLOCATE_UNIQUE ( :LOCKNAME, :LOCKHANDLE );
      END; END-EXEC.
Lock the printer in exclusive mode (default mode):

   EXEC SQL EXECUTE
      BEGIN DBMS_LOCK.REQUEST ( :LOCKHANDLE );
      END; END-EXEC.

We now have exclusive use of the printer, print the check:

   ...

Unlock the printer so other people can use it:

   EXEC SQL EXECUTE
      BEGIN DBMS_LOCK.RELEASE ( :LOCKHANDLE );

      END; END-EXEC.
```

25

DBMS_LOGMNR

Using LogMiner, you can make queries based on actual data values. For instance, you could issue a query to select all updates to the table `scott.emp` or all deletions performed by user `scott`. You could also perform a query to show all updates to `scott.emp` that increased `sal` more than a certain amount. Such data can be used to analyze system behavior and to perform auditing tasks.

The `DBMS_LOGMNR` package contains procedures used to initialize the LogMiner tool. You use these procedures to list the redo logs to be analyzed and to specify the SCN or time range of interest. After these procedures complete, the server is ready to process SQL `SELECT` statements against the `V$LOGMNR_CONTENTS` view.

> **See Also:** *Oracle9i Database Administrator's Guide* for information about using LogMiner

This chapter discusses the following topics:

DBMS_LOGMNR Constants

Table 25–1 describes the constants for the ADD_LOGFILE options flag in the DBMS_LOGMNR package.

Table 25–1 Constants for ADD_LOGFILE Options Flag

Constant	Description
NEW	DBMS_LOGMNR.NEW purges the existing list of redo logs, if any. Places the specified redo log in the list of redo logs to be analyzed.
ADDFILE	DBMS_LOGMNR.ADDFILE adds the specified redo log to the list of redo logs to be analyzed. Any attempts to add a duplicate file raise an exception (ORA-1289).
REMOVEFILE	DBMS_LOGMNR.REMOVEFILE removes the redo log from the list of redo logs to be analyzed. Any attempts to remove a file that has not been previously added, raise an exception (ORA-1290).

Table 25–2 describes the constants for the START_LOGMNR options flag in the DBMS_LOGMNR package.

Table 25–2 Constants for START_LOGMNR Options Flag

Constant	Description
COMMITTED_DATA_ONLY	If set, only DMLs corresponding to committed transactions are returned. DMLs corresponding to a committed transaction are grouped together. Transactions are returned in their commit order. If this option is not set, all rows for all transactions (committed, rolled back, and in-progress) are returned
SKIP_CORRUPTION	Directs a SELECT operation from V$LOGMNR_CONTENTS to skip any corruptions in the redo log being analyzed and continue processing. This option works only when a block in the redo log (and not the header of the redo log) has been corrupted. Caller should check the INFO column in the V$LOGMNR_CONTENTS view to determine the corrupt blocks skipped by LogMiner.
DDL_DICT_TRACKING	If the dictionary in use is a flat file or in the redo logs, LogMiner ensures that its internal dictionary is updated if a DDL event occurs. This ensures that correct SQL_REDO and SQL_UNDO information is maintained for objects that are modified after the LogMiner dictionary is built. This option cannot be used in conjunction with the DICT_FROM_ONLINE_CATALOG option.
DICT_FROM_ONLINE_CATALOG	Directs LogMiner to use the current "live" database dictionary rather than a dictionary snapshot contained in a flat file or in a redo log. This option cannot be used in conjunction with the DDL_DICT_TRACKING option.

Table 25–2 (Cont.) Constants for START_LOGMNR Options Flag

Constant	Description
DICT_FROM_REDO_LOGS	If set, LogMiner expects to find a dictionary in the redo logs that were specified with the DBMS_LOGMNR.ADD_LOGFILE procedure.
NO_SQL_DELIMITER	if set, the SQL delimiter (a semicolon) is not placed at the end of reconstructed SQL statements.
PRINT_PRETTY_SQL	If set, LogMiner formats the reconstructed SQL statements for ease of reading.
CONTINUOUS_MINE	If set, you only need to register one archived redo log. LogMiner automatically adds and mines any subsequent archived redo logs and also the online catalog. This is useful when you are mining in the same instance that is generating the redo logs.

Extracting Data Values from Redo Logs

LogMiner data extraction from redo logs is performed using two mine functions: DBMS_LOGMNR.MINE_VALUE and DBMS_LOGMNR.COLUMN_PRESENT, described later in this chapter.

Example of Using DBMS_LOGMNR

The following example shows how to use the DBMS_LOGMNR procedures to add redo logs to a LogMiner session, how to start LogMiner (with a flat file dictionary), how to perform a select operation from V$LOGMNR_CONTENTS, and how to end a LogMiner session. For complete descriptions of the DBMS_LOGMNR procedures, see Summary of DBMS_LOGMNR Subprograms on page 25-4.

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LogFileName => '/oracle/logs/log1.f', -
  3 Options => dbms_logmnr.NEW);

SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LogFileName => '/oracle/logs/log2.f', -
  3 Options => dbms_logmnr.ADDFILE);

SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -
  2 DictFileName =>'/oracle/dictionary.ora');

SQL> SELECT sql_redo
  2 FROM V$LOGMNR_CONTENTS

SQL> EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

Summary of DBMS_LOGMNR Subprograms

Table 25–3 describes the procedures in the DBMS_LOGMNR supplied package.

Table 25–3 DBMS_LOGMNR Package Subprograms

Subprogram	Description
ADD_LOGFILE Procedure on page 25-4	Adds a file to the existing or newly created list of archive files to process.
START_LOGMNR Procedure on page 25-5	Initializes the LogMiner utility.
END_LOGMNR Procedure on page 25-8	Finishes a LogMiner session.
MINE_VALUE Function on page 25-8	This function may be called for any row returned from V$LOGMNR_CONTENTS to retrieve the undo or redo column value of the column specified by the column_name input parameter to this function.
COLUMN_PRESENT Function on page 25-10	This function may be called for any row returned from V$LOGMNR_CONTENTS to determine if undo or redo column values exist for the column specified by the column_name input parameter to this function.

ADD_LOGFILE Procedure

This procedure adds a file to the existing or newly created list of archive files to process.

In order to select information from the V$LOGMNR_CONTENTS view, the LogMiner session must be set up with information about the redo logs to be analyzed. Use the ADD_LOGFILE procedure to specify the list of redo logs to analyze.

Note: If you want to analyze more than one redo log, you must call the ADD_LOGFILE procedure separately for each redo log.

Syntax

```
DBMS_LOGMNR.ADD_LOGFILE(
   LogFileName     IN VARCHAR2,
   Options         IN BINARY_INTEGER default ADDFILE );
```

Parameters

Table 25–4 describes the parameters for the ADD_LOGFILE procedure.

Table 25–4 ADD_LOGFILE Procedure Parameters

Parameter	Description
LogFileName	Name of the redo log that must be added to the list of redo logs to be analyzed by this session.
Options	Either: - Starts a new list (DBMS_LOGMNR.NEW) - Adds a file to an existing list (DBMS_LOGMNR.ADDFILE), or - Removes a redo log (DBMS_LOGMNR.REMOVEFILE) See Table 25–1, " Constants for ADD_LOGFILE Options Flag".

Exceptions

- ORA-1284: Redo log file specified cannot be opened. Log file or the directory may be non-existent or inaccessible.
- ORA-1285: Error reading the header of the redo log file.
- ORA-1286: Redo log file specified is not from the database that produced other logfiles added for analysis.
- ORA-1287: Redo log file specified is from a different database incarnation.
- ORA-1289: Redo log file specified is a duplicate of a previously specified log file.
- ORA-1290: Redo log file specified for removal is not a registered log file.
- ORA-1337: Redo log file specified has a different compatibility version than the rest of the logfiles added.

START_LOGMNR Procedure

This procedure starts a LogMiner session.

Note: This procedure fails if you did not previously use the ADD_LOGFILE procedure to specify a list of redo logs to be analyzed.

Syntax

```
DBMS_LOGMNR.START_LOGMNR(
   startScn           IN NUMBER default 0,
   endScn             IN NUMBER default 0,
   startTime          IN DATE default '01-jan-1988',
   endTime            IN DATE default '01-jan-2988',
   DictFileName       IN VARCHAR2 default '',
   Options            IN BINARY_INTEGER default 0 );
```

Parameters

Table 25–5 describes the parameters for the DBMS_LOGMNR.START_LOGMNR procedure.

Table 25–5 START_LOGMNR Procedure Parameters

Parameter	Description
startScn	Only consider redo records with SCN greater than or equal to the startSCN specified. This fails if there is no redo log with an SCN range (that is, the LOW_SCN and NEXT_SCN associated with the redo log as shown in V$LOGMNR_LOGS view) containing the startScn.
endScn	Only consider redo records with SCN less than or equal to the endSCN specified. This fails if there is no redo log with an SCN range (that is, the LOW_SCN and NEXT_SCN associated with the redo log as shown in V$LOGMNR_LOGS view) containing the endScn.
startTime	Only consider redo records with timestamp greater than or equal to the startTime specified. This fails if there is no redo log with a time range (that is, the LOW_TIME and HIGH_TIME associated with the redo log as shown in V$LOGMNR_LOGS view) containing the startTime. This parameter is ignored if startScn is specified.
endTime	Only consider redo records with timestamp less than or equal to the endTime specified. This fails if there is no redo log with a time range (that is, the LOW_TIME and HIGH_TIME associated with the redo log as shown in V$LOGMNR_LOGS view) containing the endTime. This parameter is ignored if endScn is specified.

Table 25–5 (Cont.) START_LOGMNR Procedure Parameters

Parameter	Description
DictFileName	This flat file contains a snapshot of the database catalog. It is used to reconstruct SQL_REDO and SQL_UNDO columns in V$LOGMNR_CONTENTS, as well as to fully translate SEG_NAME, SEG_OWNER, SEG_TYPE_NAME, and TABLE_SPACE columns. The fully qualified path name for the dictionary file must be specified (This file must have been created previously through the DBMS_LOGMNR_D.BUILD procedure). You only need to specify this parameter if neither DICT_FROM_REDO_LOGS nor DICT_FROM_ONLINE_CATALOG is specified.
Options	See Table 25–2, " Constants for START_LOGMNR Options Flag".

After executing the START_LOGMNR procedure, you can make use of the following views:

- V$LOGMNR_CONTENTS - contains history of information in redo logs
- V$LOGMNR_DICTIONARY - contains current information about the dictionary file
- V$LOGMNR_LOGS - contains information about the redo logs being analyzed
- V$LOGMNR_PARAMETERS - contains information about the LogMiner session

Exceptions

- ORA-1280: The procedure fails with this exception if LogMiner encounters an internal error
- ORA-1281: endScn is less than startScn
- ORA-1282: endDate is earlier than startDate
- ORA-1283: Invalid option is specified
- ORA-1292: No redo log file has been registered with LogMiner
- ORA-1293: The procedure fails with this exception for the following reasons:
  1. No logfile has (LOW_SCN, NEXT_SCN) range containing the startScn specified.
  2. No logfile has (LOW_SCN, NEXT_SCN) range containing the endScn specified.

3. No logfile has (LOW_TIME, HIGH_TIME) range containing the startTime specified.
4. No logfile has (LOW_TIME, HIGH_TIME) range containing the endTime specified.

- ORA-1294: Dictionary file specified is corrupt.
- ORA-1295: Dictionary specified does not correspond to the same database that produced the log files being analyzed.
- ORA-1296: Character set specified in the data dictionary does not match, and is incompatible with, that of the mining database.
- ORA-1297: Redo version mismatch between the dictionary and the registered redo log files.
- ORA-1299: The specified dictionary is from a different database incarnation.
- ORA-1300: Enabled thread bit vector from the dictionary does not match the redo log file. Not all redo threads have been registered with LogMiner.

END_LOGMNR Procedure

This procedure finishes a LogMiner session. Because this procedure performs cleanup operations which may not otherwise be done, you must use it to properly end a LogMiner session.

Syntax

```
DBMS_LOGMNR.END_LOGMNR;
```

Parameters

None.

Exceptions

- ORA-1307: No LogMiner session is active. The END_LOGMNR procedure was called without adding any logfiles.

MINE_VALUE Function

The MINE_VALUE function takes two arguments. The first one specifies whether to mine the redo (REDO_VALUE) or undo (UNDO_VALUE) portion of the data. The second argument is a string that specifies the fully-qualified name of the column to

be mined. The MINE_VALUE function always returns a string that can be converted back to the original datatype.

Syntax

```
dbms_logmnr.mine_value(
     sql_redo_undo      IN  RAW,
     column_name        IN  VARCHAR2 default '') RETURN VARCHAR2;
```

Parameters

Table 25–6 describes the parameters for the MINE_VALUE function.

Table 25–6 MINE_VALUE Function Parameters

Parameter	Description
sql_redo_undo	The column in V$LOGMNR_CONTENTS from which to extract data values. This parameter can be thought of as a self-describing record that contains values corresponding to several columns in a table.
column_name	Fully qualified name (schema.table.column) of the column for which this function will return information.

Returns

Table 25–7 describes the return values for the MINE_VALUE function.

Table 25–7 Return Values for MINE_VALUE Function

Return	Description
NULL	The column is not contained within the self-describing record or the column value is NULL.
NON-NULL	The column is contained within the self-describing record; the value is returned in string format.

Exceptions

- ORA-1302: Specified table or column does not exist.

Usage Notes

- To use the MINE_VALUE function, you must have successfully started a LogMiner session.

- The MINE_VALUE function must be invoked in the context of a select operation from the V$LOGMNR_CONTENTS view.
- The MINE_VALUE function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- When the column argument is of type DATE, the string that is returned is formatted in canonical form (DD-MON-YYYY HH24:MI:SS.SS) regardless of the date format of the current session.

COLUMN_PRESENT Function

This function is meant to be used in conjunction with the MINE_VALUE function.

If the MINE_VALUE function returns a NULL value, it can mean either:

- The specified column is not present in the redo or undo portion of the data.
- The specified column is present and has a null value.

To distinguish between these two cases, use the COLUMN_PRESENT function which returns a 1 if the column is present in the redo or undo portion of the data. Otherwise, it returns a 0.

Syntax

```
dbms_logmnr.column_present(
     sql_redo_undo      IN  RAW,
     column_name        IN  VARCHAR2 default '') RETURN NUMBER;
```

Parameters

Table 25–8 describes the parameters for the COLUMN_PRESENT function.

Table 25–8 COLUMN_PRESENT Function Parameters

Parameter	Description
sql_redo_undo	The column in V$LOGMNR_CONTENTS from which to extract data values. This parameter can be thought of as a self-describing record that contains values corresponding to several columns in a table.
column_name	Fully qualified name (schema.table.column) of the column for which this function will return information.

Returns

Table 25–9 describes the return values for the COLUMN_PRESENT function.

Table 25–9 Return Values for COLUMN_PRESENT Function

Return	Description
0	Specified column is not present in this row of V$LOGMNR_CONTENTS.
1	Column is present in this row of V$LOGMNR_CONTENTS. Returns 1 if the self-describing record (the first parameter) contains the column specified in the second parameter. This can be used to distinguish between NULL returns from the DBMS_LOGMNR.MINE_VALUE function.

Exceptions

- ORA-1302: Specified table or column does not exist.

Usage Notes

- To use the COLUMN_PRESENT function, you must have successfully started a LogMiner session.
- The COLUMN_PRESENT function must be invoked in the context of a select operation from the V$LOGMNR_CONTENTS view.
- The COLUMN_PRESENT function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- When the column argument is of type DATE, the string that is returned is formatted in canonical form (DD-MON-YYYY HH24:MI:SS.SS) regardless of the date format of the current session.

26

DBMS_LOGMNR_CDC_PUBLISH

Oracle Change Data Capture identifies new data that has been added to, modified, or removed from relational tables and publishes the changed data in a form that is usable by an application.

This chapter describes how to use the `DBMS_LOGMNR_CDC_PUBLISH` supplied package to set up an Oracle Change Data Capture system to capture and publish data from one or more Oracle relational source tables. Change Data Capture captures and publishes only committed data.

Typically, a Change Data Capture system has one publisher that captures and publishes changes for any number of Oracle source (relational) tables. The publisher then provides subscribers, typically applications, with access to the published data.

> **See Also:** *Oracle9i Data Warehousing Guide* for more information about the Oracle Change Data Capture publish and subscribe model.

This chapter discusses the following topics:

Publishing Change Data

The publisher, typically a database administrator, is concerned primarily with the source of the data and with creating the schema objects that describe the structure of the capture system: change sources, change sets, and change tables.

Most Change Data Capture systems have one publisher and many subscribers. The publisher accomplishes the following main objectives:

1. Determine which source table changes need to be published.
2. Use the procedures in the DBMS_LOGMNR_CDC_PUBLISH package to capture change data and makes it available from the source tables by creating and administering the change source, change set, and change table objects.
3. Allow controlled access to subscribers by using the SQL GRANT and REVOKE statements to grant and revoke the SELECT privilege on change tables for users and roles.

   This is necessary to allow the subscribers, usually applications, to use the DBMS_LOGMNR_CDC_SUBSCRIBE procedure to subscribe to the change data.

Summary of DBMS_LOGMNR_CDC_PUBLISH Subprograms

Through the DBMS_LOGMNR_CDC_PUBLISH package, the publisher creates and maintains change sources, change sets, and change tables, and eventually drops them when they are no longer useful.

Note: To use the DBMS_LOGMNR_CDC_PUBLISH package, you must have the EXECUTE_CATALOG_ROLE privilege, and you must have the SELECT_CATALOG_ROLE privilege to look at all of the views.

Table 26–1 describes the procedures in the DBMS_LOGMNR_CDC_PUBLISH supplied package.

Table 26–1 DBMS_LOGMNR_CDC_PUBLISH Package Subprograms

Subprogram	Description
CREATE_CHANGE_TABLE Procedure on page 26-3	Creates a change table in a specified schema and creates corresponding Change Data Capture metadata.
ALTER_CHANGE_TABLE Procedure on page 26-8	Adds or drops columns for an existing change table, or changes the properties of an existing change table.
DROP_SUBSCRIBER_VIEW Procedure on page 26-12	Allows the publisher to drop a subscriber view from the subscriber's schema. The view must have been created by a prior call to the PREPARE_SUBSCRIBER_VIEW procedure.
DROP_SUBSCRIPTION Procedure on page 26-13	Allows a publisher to drop a subscription that was created with a prior call to the GET_SUBSCRIPTION_HANDLE procedure.
DROP_CHANGE_TABLE Procedure on page 26-14	Drops an existing change table when there is no more activity on the table.
PURGE Procedure on page 26-16	Monitors usage by all subscriptions, determines which rows are no longer needed by subscriptions, and removes the unneeded rows to prevent change tables from growing endlessly.

CREATE_CHANGE_TABLE Procedure

This procedure creates a change table in a specified schema.

Syntax

The following syntax specifies columns and datatypes using a comma-delimited string.

```
DBMS_LOGMNR_CDC_PUBLISH.CREATE_CHANGE_TABLE (
     owner                  IN VARCHAR2,
     change_table_name      IN VARCHAR2,
     change_set_name        IN VARCHAR2,
     source_schema          IN VARCHAR2,
     source_table           IN VARCHAR2,
     column_type_list       IN VARCHAR2,
     capture_values         IN VARCHAR2,
     rs_id                  IN CHAR,
     row_id                 IN CHAR,
     user_id                IN CHAR,
     timestamp              IN CHAR,
     object_id              IN CHAR,
     source_colmap          IN CHAR,
     target_colmap          IN CHAR,
     options_string         IN VARCHAR2);
```

Parameters

Table 26–2 CREATE_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	Name of the schema that owns the change table.
change_table_name	Name of the change table that is being created.
change_set_name	Name of an existing change set with which this change table is associated. Synchronous change tables must specify SYNC_SET.
source_schema	The schema where the source table is located.
source_table	The source table from which the change records are captured.
column_type_list	Comma-delimited list of columns and datatypes that are being tracked.
capture_values	Set this parameter to one of the following capture values for update operations: ▪ OLD: Captures the original values from the source table. ▪ NEW: Captures the changed values from the source table. ▪ BOTH: Captures the original and changed values from the source table.
rs_id	Adds a column to the change table that contains the row sequence number. This parameter orders the operations in a transaction in the sequence that they were committed in the database. The row sequence ID (rs_id) parameter is optional for synchronous mode. **Note:** For synchronous mode, the rs_id parameter reflects an operations capture order within a transaction, but you cannot use the rs_id parameter by itself to order committed operations across transactions. Set this parameter to Y or N, as follows: Y: Indicates that you want to add a column to the change table that will contain the row sequence of the change. N: Indicates that you do not want to track the rs_id column.
row_id	Adds a column to the change table that contains the row ID of the changed row in the source table, as follows. Y: Indicates that you want to add a column to the change table that contains the row ID of the changed row in the source table. N: Indicates that you do not want to track the row_id column.

Table 26–2 CREATE_CHANGE_TABLE Procedure Parameters

Parameter	Description
`user_id`	Adds a column to the change table that contains the user name of the user who entered a DML statement, as follows. Y: Indicates that you want to add a column to the change table that contains the user name of the user who entered a DML statement. N: Indicates that you do not want to track users.
`timestamp`	Adds a column to the change table that contains the capture timestamp of the change record, as follows: Y: Indicates that you want to add a column to the change table that contains the capture timestamp of the change record. N: Indicates that you do not want to track timestamps.
`object_id`	Adds a column to the change table that contains the object ID of this change record. This is a control column for object support. Specify Y or N, as follows: Y: Indicates that you want to add a column to the change table that contains the object ID of this change record. N: Indicates that you do not want to track object IDs.
`source_colmap`	Adds a column to the change table as a change column vector that indicates which source columns actually changed. Specify Y or N, as follows: Y: Indicates that you want to add a column to the change table to track the source columns that have changed. N: Indicates that you do not want to track which source columns changed.
`target_colmap`	Adds a column to the change table as a column vector indicating which change table user columns actually changed. Specify Y or N, as follows. Y: Indicates that you want to add a column to the change table to track the change table user columns that have changed. N: Indicates that you do not want to track changes which change table user columns changed.
`options_string`	A string that contains syntactically correct options to be passed to a `CREATE TABLE` DDL statement. The options string is appended to the generated `CREATE TABLE` DDL statement after the closing parenthesis that defines the columns of the table. See the Usage Notes for more information.

Exceptions

Table 26–3 CREATE_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31409	One or more of the input parameters to the CREATE_CHANGE_TABLE procedure had invalid values. Identify the incorrect parameters and supply the correct values to the procedure.
ORA-31416	The value specified for the source_colmap parameter is invalid. For synchronous mode, specify either Y or N.
ORA-31417	A reserved column name was specified in a column list or column type parameter. Ensure that the name specified does not conflict with a reserved column name.
ORA-31418	While creating a synchronous change table, the name of the source schema did not match any existing schema name in the database.
ORA-31419	When creating a synchronous change table, the underlying source table did not exist when the procedure was called.
ORA-31420	When creating the first change table, a purge job is submitted to the job queue. Submission of this purge job failed.
ORA-31421	The specified change table does not exist. Check the specified change table name to see that it matches the name of an existing change table.
ORA-31422	Owner schema does not exist.
ORA-31438	Duplicate change table. Re-create the change table with a unique name.
ORA-31450	Invalid value was specified for change_table_name.
ORA-31451	Invalid value was specified for the capture_value. Expecting either OLD, NEW, or BOTH.
ORA-31452	Invalid value was specified. Expecting either Y or N.
ORA-31459	System triggers for DBMS_LOGMRN_CDC_PUBLISH package are not installed.
ORA-31467	No column found in the source table. The OBJECT_ID flag was set to Y on the call to CREATE_CHANGE_TABLE and change table belongs to the synchronous change set. The corresponding object column was not detected in the source table.

Usage Notes

- A change table is a database object that contains the change data resulting from DML statements (INSERT, UPDATE, and DELETE) made to a source table. A given change table can capture changes from only one source table.
- A synchronous change table must belong to the SYNC_SET change set.
- A change table is a database table that maintains the change data in these two types of columns:
  - Source columns identify the columns from the source table to capture. Source columns are copies of actual source table columns that reside in the change table.
  - Control columns maintain special metadata for each change row in the container table. Information such as the DML operation performed, the capture time (timestamp), and changed column vectors are examples of control columns.
- The publisher can control a change table's physical properties, tablespace properties, and so on by specifying the options_string parameter. With the options_string parameter, you can set any option that is valid for the CREATE TABLE DDL statement.
- Do not attempt to control a change table's partitioning properties. When Change Data Capture performs a purge operation to remove rows from a change set, it automatically manages the change table partitioning for you.

> **Note:** How you define the options_string parameter can have an effect on the performance and operations in a Change Data Capture system. For example, if the publisher places several constraints in the options column, it can have a noticeable effect on performance. Also, if the publisher uses NOT NULL constraints and a particular column is not changed in an incoming change row, then the constraint can cause the entire INSERT operation to fail.

Example

```
execute DBMS_CDC_PUBLISH.CREATE_CHANGE_TABLE(OWNER => 'cdc1', \
   CHANGE_TABLE_NAME => 'emp_ct', \
   CHANGE_SET_NAME => 'SYNC_SET', \
   SOURCE_SCHEMA => 'scott', \
   SOURCE_TABLE => 'emp', \
```

```
   COLUMN_TYPE_LIST => 'empno number, ename varchar2(10), job varchar2(9), mgr
number, hiredate date, deptno number', \
   CAPTURE_VALUES => 'both', \
   RS_ID => 'y', \
   ROW_ID => 'n', \
   USER_ID => 'n', \
   TIMESTAMP => 'n', \
   OBJECT_ID => 'n',\
   SOURCE_COLMAP => 'n', \
   TARGET_COLMAP => 'y', \
   OPTIONS_STRING => NULL);
```

ALTER_CHANGE_TABLE Procedure

This procedure adds columns to, or drops columns from, an existing change table.

Syntax

The following syntax specifies columns and datatypes as a comma-delimited list.

```
DBMS_LOGMNR_CDC_PUBLISH.ALTER_CHANGE_TABLE (
     owner                  IN VARCHAR2,
     change_table_name      IN VARCHAR2,
     operation              IN VARCHAR2,
     column_list            IN VARCHAR2,
     rs_id                  IN CHAR,
     row_id                 IN CHAR,
     user_id                IN CHAR,
     timestamp              IN CHAR,
     object_id              IN CHAR,
     source_colmap          IN CHAR,
     target_colmap          IN CHAR);
```

Parameters

Table 26–4 ALTER_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	Name of the schema that owns the change table.
change_table_name	Name of the change table that is being altered.
operation	Specifies either the value DROP or ADD to indicate whether to add or drop the columns in the field column_table or column_list.

Table 26–4 ALTER_CHANGE_TABLE Procedure Parameters

Parameter	Description
column_list	A comma-delimited list of column names and datatypes for each column of the source table that should be added to, or dropped from, the change table.
rs_id	Adds or drops the control column that tracks the row sequence (rs_id). Set this parameter to Y or N, as follows: Y: Adds or drops a column on the change table that contains the row sequence (rs_id). N: The rs_id control column is not changed in the change table.
row_id	Adds or drops a row_id column, as follows: Y: Adds or drops the row_id control column for the change table. N: The row_id column is not changed in the change table.
user_id	Adds or drops the user name control column. Specify Y or N, as follows: Y: Adds or drops a column on the change table that contains the user name (user_id). N: The user_id column is not changed in the change table.
timestamp	Adds or drops the timestamp control column to the change table, as follows: Y: Adds or drops a column on the change table that contains the timestamp. N: The timestamp control column is not changed in the change table.
object_id	Add or drops the object_id column, as follows: Y: Adds or drops a column on the change table that contains the object_id. N: The object_id control column is not changed in the change table.
source_colmap	Adds or drops the source_colmap control column from the change table, as follows: Y: Adds or drops a column on the change table that contains the source columns (source_colmap). N: The source_colmap column is not changed in the change table.

Table 26–4 ALTER_CHANGE_TABLE Procedure Parameters

Parameter	Description
target_colmap	Adds or drops the target_colmap control column from the change table, as follows: Y: Adds or drops a column on the change table that contains the target columns (target_colmap). N: The target_colmap column is not changed in the change table.

Exceptions

Table 26–5 ALTER_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31403	You issued an ALTER_CHANGE_TABLE procedure with an ADD operation but a column by this name already exists in the specified table.
ORA-31409	One or more of the input parameters to the ALTER_CHANGE_SET procedure had invalid values. Identify the incorrect parameters and supply the correct values to the procedure.
ORA-31417	A reserved column name was specified in the column list parameter. Ensure that the name specified does not conflict with a reserved column name.
ORA-31421	The specified change table does not exist. Check the specified change table name to see that it matches the name of an existing change table.
ORA-31423	You issued the ALTER_CHANGE_TABLE with a drop operation and the specified column does not exist in the change table.
ORA-31454	Illegal value was specified for operation parameter; expecting ADD or DROP.
ORA-31455	Nothing to alter. The specified column list is NULL and all optional control columns are N.
ORA-31456	An internal attempt to invoke a procedure within the DBMS_CDC_UTILITY package failed. Check the trace logs for more information.
ORA-31459	One or more required system triggers are not installed.

Usage Notes

- You cannot add and drop user columns in the same call to the ALTER_CHANGE_TABLE procedure; these schema changes require separate calls.
- Do not specify the name of the control columns in the user-column lists.

- The following table describes what happens when you add a column to a change table:

If the publisher adds	And	Then . . .
A user column	A new subscription includes this column	The subscription window starts at the point the column was added.
A user column	A new subscription does not include this newly added column	The subscription window starts at the low-water mark for the change table thus enabling the subscriber to see the entire table.
A user column	Old subscriptions exist	The subscription window remains unchanged and the entire table can be seen.
A control column	There is a new subscription	The subscription window starts at the low-water mark for the change table. The subscription can see the control column immediately. All rows that existed in the change table prior to adding the control column will have the value NULL for the newly added control column field.
A control column	—	Any existing subscriptions can see the new control column when the window is extended (DBMS_LOGMNR_CDC_PUBLISH.EXTEND_WINDOW procedure) such that the low watermark for the window crosses over the point when the control column was added.

Example

```
EXECUTE DBMS_LOGMNR_CDC_PUBLISH.ALTER_CHANGE_TABLE (OWNER => 'cdc1') \
   CHANGE_TABLE_NAME => 'emp_ct' \
   OPERATION => ADD \
   ADD_COLUMN_LIST => '' \
   RS_ID => 'Y' \
   ROW_ID => 'N' \
   USER_ID => 'N' \
   TIMESTAMP => 'N' \
   OBJECT_ID => 'N' \
```

```
    SOURCE_COLMAP => 'N' \
    TARGET_COLMAP => 'N');
```

DROP_SUBSCRIBER_VIEW Procedure

This procedure allows a publisher to drop a subscriber view in the subscriber's schema.

> **Note:** This procedure works the same way as the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW procedure.

Syntax

```
DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIBER_VIEW (
     subscription_handle    IN NUMBER,
     source_schema          IN VARCHAR2,
     source_table           IN VARCHAR2)
```

Parameters

Table 26–6 DROP_SUBSCRIBER_VIEW Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE procedure.
source_schema	Schema name where the source table resides.
source_table	Name of the published source table.

Exceptions

Table 26–7 DROP_SUBSCRIBER_VIEW Procedure Exceptions

Exception	Description
ORA-31425	Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.
ORA-31429	The subscription has not been activated. Check the subscription handle and correct it, if necessary. Call the DBMS_LOGMNR_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION procedure for this subscription handle and then try the original command again.

Table 26–7 DROP_SUBSCRIBER_VIEW Procedure Exceptions

Exception	Description
ORA-31432	The schema_name.source_table does not exist or does not belong to this subscription. Check the spelling of the schema_name and source_table parameters. Verify the specified table exists in the specified schema and is subscribed to by the subscription handle.
ORA-31433	The subscriber view does not exist. Either you specified an incorrect subscriber view or the view is already dropped. Check the name and specify the name of an existing subscriber view.

Usage Notes

- This procedure provides the publisher with a way to clean up views that have not been removed by the subscriber. (Typically, subscribers drop the subscriber views using the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW procedure.)
- The subscriber view you want to drop must have been created with a prior call to the DBMS_LOGMNR_CDC_SUBSCRIBE.PREPARE_SUBSCRIBER_VIEW procedure.
- You must use this procedure to drop any subscriber views prior to dropping a subscription using the DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIPTION procedure.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW( \
   SUBSCRIPTION_HANDLE =>:subhandle, \
   SOURCE_SCHEMA =>'scott', \
   SOURCE_TABLE => 'emp');
```

DROP_SUBSCRIPTION Procedure

This procedure allows a publisher to drop a subscription that was created with a prior call to the DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE procedure.

> **Note:** This procedure works the same way as the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure.

Syntax

```
DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIPTION (
     subscription_handle  IN NUMBER)
```

Parameters

Table 26–8 DROP_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE procedure.

Exceptions

Table 26–9 DROP_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31425	Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.
ORA-31430	The subscriber view was not dropped prior to making this call. Call the DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIBER_VIEW procedure and then try the original command again.

Usage Notes

- This procedure provides the publisher with a way to drop subscriptions that have not been dropped by the subscriber. (Typically, subscribers drop subscriptions using the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure.)
- Prior to dropping a subscription, you must drop the subscriber view using the DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIBER_VIEW procedure.

Example

```
EXECUTE DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIPTION ( \
   SUBSCRIPTION_HANDLE => :subhandle);
```

DROP_CHANGE_TABLE Procedure

This procedure drops an existing change table.

Syntax

```
DBMS_LOGMNR_CDC_PUBLISH.DROP_CHANGE_TABLE (
     owner              IN VARCHAR2,
     change_table_name  IN VARCHAR2,
     force_flag         IN CHAR)
```

Parameters

Table 26–10 DROP_CHANGE_TABLE Procedure Parameters

Parameter	Description
owner	Name of the schema that owns the change table.
change_table_name	Name of the change table that is being dropped.
force_flag	Drops the change table, depending on whether or not there are subscriptions making references to it, as follows: Y: Drops the change table even if there are subscriptions making references to it. N: Drops the change table only if there are no subscribers referencing it.

Exceptions

Table 26–11 DROP_CHANGE_TABLE Procedure Exceptions

Exception	Description
ORA-31421	The specified change table does not exist. Check the specified change table name to see that it matches the name of an existing change table.
ORA-31422	Owner schema does not exist.
ORA-31424	The specified change table has active subscriptions, and thus it cannot be dropped. If you must drop the table, use the `force_flag` parameter to immediately drop the change table from all of the subscribers.
ORA-31441	Table is not a change table. You attempted to execute the DROP_CHANGE_TABLE procedure on a table that is not a change table.

Example

```
EXECUTE DBMS_LOGMNR_CDC_PUBLISH.DROP_CHANGE_TABLE ( \
   OWNER => 'cdc1', \
   CHANGE_TABLE_NAME => 'emp_ct' \
   FORCE_FLAG => 'N')
```

PURGE Procedure

This procedure monitors change table usage by all subscriptions, determines which rows are no longer needed by subscriptions, and removes the unneeded rows to prevent change tables from growing endlessly.

Syntax

```
DBMS_LOGMNR_CDC_PUBLISH.PURGE ( )
```

Exceptions

Only standard Oracle exceptions (for example, a privilege violation) are returned during a purge operation.

Usage Notes

- You can run this procedure manually or automatically:
  - Run this procedure manually from the command line at any time that you want to purge data from change tables.
  - Run this procedure in a script to routinely perform a purge operation and proactively control the growth of change tables. You can always remove or disable (or suspend) the purge operation if you want to prevent it from running automatically.
- Use this procedure to control the growth of change tables.
- Do not attempt to control a change table's partitioning properties. When the `DBMS_LOGMNR_CDC_PUBLISH.PURGE` procedure runs, Change Data Capture performs partition maintenance automatically.

Example

```
EXECUTE DBMS_LOGMNR_CDC_PUBLISH.PURGE
```

27

DBMS_LOGMNR_CDC_SUBSCRIBE

This chapter describes how to use the DBMS_LOGMNR_CDC_SUBSCRIBE package to view and query the change data that was captured and published with the DBMS_LOGMNR_CDC_PUBLISH package.

A Change Data Capture system usually has one publisher that captures and publishes changes for any number of Oracle source (relational) tables and many subscribers. The subscribers, typically applications, use the Oracle supplied package, DBMS_LOGMNR_CDC_SUBSCRIBE, to access the published data.

> **See Also:** *Oracle9i Data Warehousing Guide* for more information about the Oracle Change Data Capture publish and subscribe model.

This chapter discusses the following topics:

Subscribing to Change Data

Once the publisher sets up the system to capture data into change tables and grants access, subscribers can access and query the published change data for any of the source tables of interest. Using the procedures in the DBMS_LOGMNR_CDC_SUBSCRIBE package, the subscriber accomplishes the following main objectives:

1. Indicate the change data of interest by creating subscriptions to published source tables and source columns.
2. Extend the subscription window and create a new subscriber view when the subscriber is ready to receive a set of change data.
3. Use SELECT statements to retrieve change data from the subscriber views.
4. Drop the subscriber view and purge the subscription window when finished processing a block of changes.
5. Drop the subscription when the subscriber no longer needs its change data.

Summary of DBMS_LOGMNR_CDC_SUBSCRIBE Subprograms

The primary role of the subscriber is to use the change data. Through the DBMS_LOGMNR_CDC_SUBSCRIBE package, each subscriber registers interest in a set of source tables by *subscribing* to them.

Table 27–1 describes the procedures for the DBMS_LOGMNR_CDC_SUBSCRIBE package.

Table 27–1 DBMS_LOGMNR_CDC_SUBSCRIBE Package Subprograms

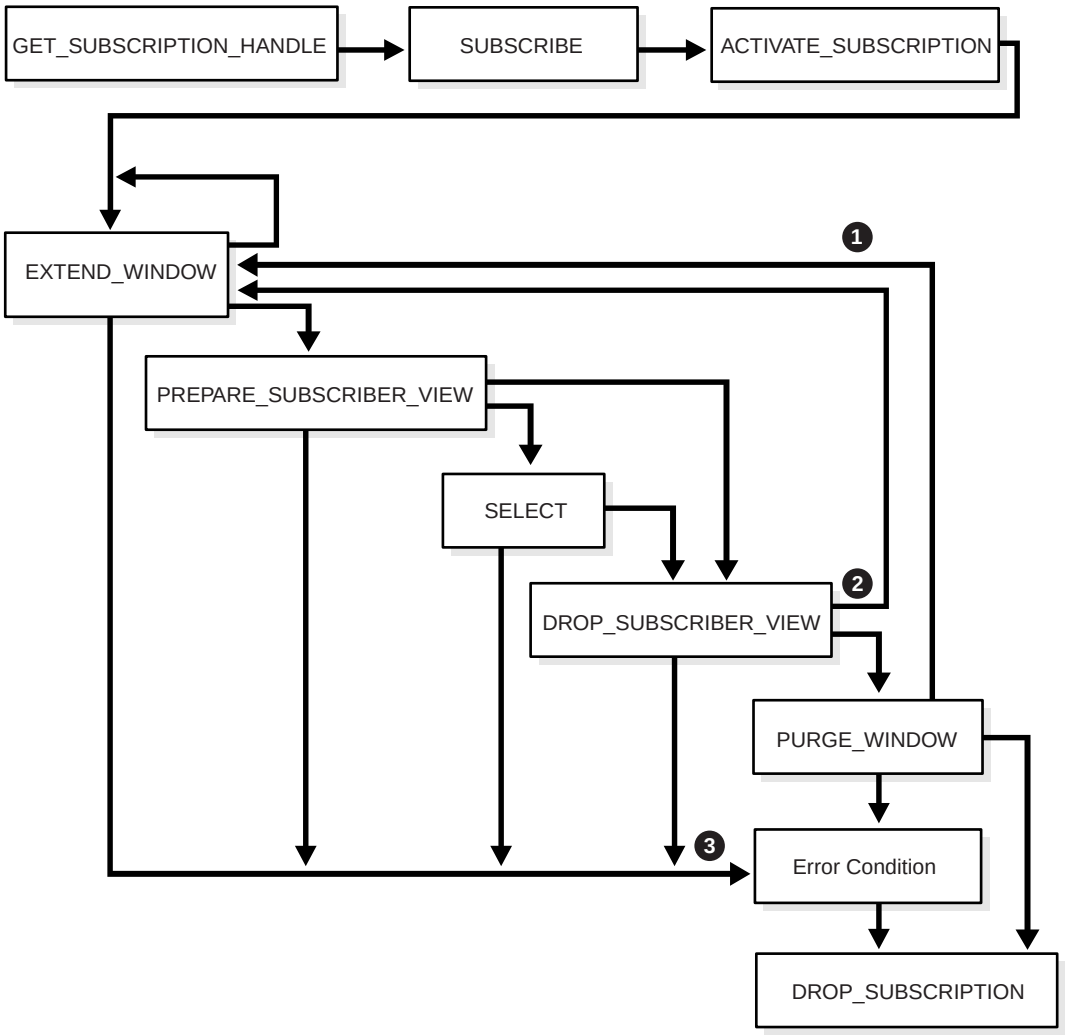
Subprogram	Description
GET_SUBSCRIPTION_HANDLE Procedure on page 27-5	Creates a subscription handle that associates the subscription with one change set.
SUBSCRIBE Procedure on page 27-6	Specifies the source tables and source columns for which the subscriber wants to access change data.
ACTIVATE_SUBSCRIPTION Procedure on page 27-9	Indicates that a subscription is ready to start accessing change data.
EXTEND_WINDOW Procedure on page 27-10	Sets the subscription window boundaries (low-water and high-water mark) so that new change data can be seen.
PREPARE_SUBSCRIBER_VIEW Procedure on page 27-11	Creates a subscriber view in the subscriber's schema in which the subscriber can query the change data encompassed by the current subscription window.

Table 27–1 DBMS_LOGMNR_CDC_SUBSCRIBE Package Subprograms (Cont.)

Subprogram	Description
DROP_SUBSCRIBER_VIEW Procedure on page 27-13	Drops a subscriber view from the subscriber's schema.
PURGE_WINDOW Procedure on page 27-14	Sets the low-water mark for a subscription window to notify the capture system that the subscriber is finished processing a set of change data.
DROP_SUBSCRIPTION Procedure on page 27-14	Drops a subscription that was created with a prior call to the GET_SUBSCRIPTION_HANDLE procedure.

Subscribers call the procedures in the order shown in Table 27–1 unless an error occurs, at which time the subscribers should exit. Figure 27–1 shows the most common steps for using the procedures in the DBMS_LOGMNR_CDC_SUBSCRIBE package.

Figure 27–1 Subscription Flow



In Figure 27–1:

1. If you use the `PURGE_WINDOW` procedure immediately after using an `EXTEND_WINDOW` procedure, then change data is lost without ever being processed.

2. If you use the EXTEND_WINDOW procedure immediately after using the DROP_SUBSCRIBER_VIEW procedure, you will see the data that you just processed again and possibly some new data.
3. If an error occurs during any step in the process, the application program calling the DBMS_LOGMNR_CDC_SUBSCRIBE procedures should detect the error and exit. For example, if the PREPARE_SUBSCRIBER_VIEW procedure fails for any reason, and the application ignores the error and continues, then the PURGE_WINDOW procedure will delete data that was never seen or selected by the subscriber.

GET_SUBSCRIPTION_HANDLE Procedure

This procedure creates a subscription handle that associates the subscription with one change set. Creating a subscription handle is the first step in obtaining a subscription.

Syntax

```
DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE(
   change_set           IN  VARCHAR2,
   description          IN  VARCHAR2 := NULL,
   subscription_handle  OUT NUMBER);
```

Parameters

Table 27–2 GET_SUBSCRIPTION_HANDLE Procedure Parameters

Parameter	Description
change_set	Name of an existing change set to which the application subscribes. You must set the value to SYNC_SET.
description	Describes the subscription handle and the purpose for which it is used.
subscription_handle	Unique number of the subscription handle for this subscription.

Exception

Table 27–3 GET_SUBSCRIPTION_HANDLE Procedure Exceptions

Exception	Description
ORA-31415	Could not find an existing change set with this name.

Table 27–3 GET_SUBSCRIPTION_HANDLE Procedure Exceptions (Cont.)

Exception	Description
ORA-31457	The maximum number of characters permitted in the description field was exceeded.
ORA-31458	This is an internal error. Contact Oracle Support Services and report the error.

Usage Notes

- The GET_SUBSCRIPTION_HANDLE procedure allows a subscriber to register interest in a change set associated with source tables of interest.
- To see all of the published source tables for which the subscriber has privileges, query the ALL_PUBLICATIONS view.
- A subscriber can later use a single subscription handle to access the multiple change tables in the subscription.
- Subscription handles:
  - Never get reused and are tracked from the time of creation until they are dropped with the DROP_SUBSCRIPTION procedure.
  - Are not shared among subscribers; rather, each subscription handle is validated against the subscriber's login ID.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE(\
   CHANGE_SET=>'SYNC_SET', \
   DESCRIPTION=>'Change data for emp',\
   SUBSCRIPTION_HANDLE=>:subhandle);
```

SUBSCRIBE Procedure

This procedure specifies the source tables and source columns for which the subscriber wants to access change data.

Syntax

There are two versions of syntax for the SUBSCRIBE procedure, each of which specifies the subscriber columns and datatypes. If the subscribers know which publication contains the source columns of interest, the subscribers can use the version of the procedure that contains the publication ID. If they do not know the

publication ID, the Change Data Capture system will select a publication based on the supplied source schema and source table.

The following syntax identifies the source table of interest, allowing Change Data Capture to select any publication that contains all source columns of interest.

```
DBMS_LOGMNR_CDC_SUBSCRIBE.SUBSCRIBE (
   subscription_handle  IN NUMBER,
   source_schema        IN VARCHAR2,
   source_table         IN VARCHAR2,
   column_list          IN VARCHAR2);
```

The following syntax specifies the publication ID for a specific publication that contains the source columns of interest.

```
DBMS_LOGMNR_CDC_SUBSCRIBE.SUBSCRIBE (
   subscription_handle  IN NUMBER,
   publication_id       IN NUMBER,
   column_list          IN VARCHAR2);
```

Parameters

Table 27–4 SUBSCRIBE Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.
source_schema	Schema name where the source table resides.
source_table	Name of a published source table.
column_list	A comma-delimited list of columns from the published source table.
publication_id	A valid publication_id, which you can obtain from the ALL_PUBLISHED_COLUMNS view.

Exceptions

Table 27–5 SUBSCRIBE Procedure Exceptions

Exception	Description
ORA-31425	The specified subscription handle does not exist, or it does not belong to this user or application.

Table 27–5 SUBSCRIBE Procedure Exceptions (Cont.)

Exception	Description
ORA-31426	The subscription handle has been activated; additional calls to the SUBSCRIBE procedure are prohibited. You must subscribe to all of the desired tables and columns before activating the subscription. Ensure that the correct subscription handle was specified.
ORA-31427	The subscription represented by the subscription handle already contains the schema name and source table. Check the values of the subscription_handle, source_schema, and source_table parameters. Do not attempt to subscribe to the same table more than once using the same subscription handle.
ORA-31428	No publication contains all of the specified columns. One or more of the specified columns cannot be found in a single publication. Consult the ALL_PUBLISHED_COLUMNS view to see the current publications and change the subscription request to select only the columns that are in the same publication.

Usage Notes

- You can subscribe to any valid publication_id. You can find valid publications in the ALL_PUBLISHED_COLUMNS view.
- The SUBSCRIBE procedure allows an application to subscribe to one or more published source tables and to specific columns in each source table.
- To see all of the published source table columns for which the subscriber has privileges, query the ALL_PUBLISHED_COLUMNS view.
- Subscriptions must be created before the application actually needs the data. The Change Data Capture system does not guarantee that there will be any change data available at the moment the subscription is created.
- Subscribers can subscribe only to published columns from the source table. Also, all of the columns must come from the same publication. Any control columns associated with the underlying change table are added to the subscription automatically.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.SUBSCRIBE(\
   SUBSCRIPTION_HANDLE=>:subhandle, \
   SOURCE_SCHEMA=>'scott', \
   SOURCE_TABLE=>'emp', \
   COLUMN_LIST=>'empno, ename, hiredate');
```

ACTIVATE_SUBSCRIPTION Procedure

The ACTIVATE_SUBSCRIPTION procedure indicates that a subscription is ready to start accessing change data.

Syntax

```
DBMS_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION (
   subscription_handle  IN NUMBER);
```

Parameters

Table 27–6 ACTIVATE_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.

Exceptions

Table 27–7 ACTIVATE_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31425	The specified subscription handle does not exist, or it does not belong to this user ID or application.
ORA-31439	The subscription is already active. You can activate a subscription only once.

Usage Notes

- The ACTIVATE_SUBSCRIPTION procedure indicates that you are finished subscribing to tables, and the subscription is ready to start accessing data.
- Once the subscriber activates the subscription:
  - No additional source tables can be added to the subscription.
  - The Change Data Capture system holds the available data for the source tables and sets the subscription window to empty.
  - The subscriber must use the EXTEND_WINDOW procedure to see the initial set of change data.
  - The subscription cannot be activated again.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION( \
   SUBSCRIPTION_HANDLE=>:subhandle);
```

EXTEND_WINDOW Procedure

This procedure sets the subscription window boundaries (low-water and high-water mark) so that new change data can be seen.

Syntax

```
DBMS_LOGMNR_CDC_SUBSCRIBE.EXTEND_WINDOW (
   subscription_handle  IN NUMBER);
```

Parameters

Table 27–8 EXTEND_WINDOW Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.

Exceptions

Table 27–9 EXTEND_WINDOW Procedure Exceptions

Exception	Description
ORA-31425	The specified subscription handle does not exist or it does not belong to this user or application.
ORA-31429	The subscription handle must be activated before you use the EXTEND_WINDOW procedure. Call the ACTIVATE_SUBSCRIPTION procedure for this subscription handle and then try the original command again.
ORA-31430	The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW procedure and then try the original command again.

Usage Notes

- Until you call the EXTEND_WINDOW procedure to begin capturing change data, the subscription window remains empty.

- The first time that you call the EXTEND_WINDOW procedure, it establishes the initial boundaries for the subscription window.
- Subsequent calls to the EXTEND_WINDOW procedure extend the high-water mark of the subscription window so that new change data can be seen.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.EXTEND_WINDOW( \
subscription_handle=>:subhandle);
```

PREPARE_SUBSCRIBER_VIEW Procedure

This procedure creates a subscriber view in the subscriber's schema in which the subscriber can query the change data encompassed by the current subscription window.

Syntax

```
DBMS_LOGMNR_CDC_SUBSCRIBE.PREPARE_SUBSCRIBER_VIEW (
   subscription_handle  IN  NUMBER,
   source_schema        IN  VARCHAR2,
   source_table         IN  VARCHAR2,
   view_name            OUT VARCHAR2);
```

Parameters

Table 27–10 PREPARE_SUBSCRIBER_VIEW Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.
source_schema	Schema name where the source table resides.
source_table	Name of the published source table that belongs to the subscription handle.
view_name	Name of the newly-created view that will return the change data for the source table.

Exceptions

Table 27–11 PREPARE_SUBSCRIBER_VIEW Procedure Exceptions

Exception	Description
ORA-31425	The specified subscription handle does not exist, or it does not belong to this user or application.
ORA-31429	The subscription has not been activated. The subscription handle must be activated before you use the PREPARE_SUBSCRIBER_VIEW procedure. Call the ACTIVATE_SUBSCRIPTION procedure for this subscription handle and then try the original command again.
ORA-31430	An earlier subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW procedure and then try the original command again.
ORA-31432	The schema name or source table does not exist or does not belong to this subscription. Check the spelling of the schema_name and source_table parameters. Verify the specified table exists in the specified schema and is subscribed to by the subscription handle.

Usage Notes

- This procedure creates a subscriber view in the subscriber's schema in which to display the change data. After the subscriber view is created, the subscriber can select change data that is within the boundaries defined (by the EXTEND_WINDOW procedure) for the subscription window.
- The Change Data Capture system determines the name of the subscriber view and returns the name to the subscriber. The name of the subscriber view is constant over the life of the subscription. To access the change data, there must be a view for each source table in the subscription. Applications use a SELECT statement from these views and retrieve the change data. For the purpose of the following example, assume that sys.sub9view was the view name returned by the PREPARE_SUBSCRIBER_VIEW procedure:

```
SELECT * FROM sys.sub9view;
.
.
.
```

- If a view already exists with the same view_name (for example, if the previous view was not dropped with a DROP VIEW DDL statement), an exception occurs. The PREPARE_SUBSCRIBER_VIEW procedure checks if the underlying change table still exists.

Examples

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.PREPARE_SUBSCRIBER_VIEW( \
   SUBSCRIPTION_HANDLE =>:subhandle, \
   SOURCE_SCHEMA =>'scott', \
   SOURCE_TABLE => 'emp', \
   VIEW_NAME => :viewname);
```

DROP_SUBSCRIBER_VIEW Procedure

This procedure drops a subscriber view from the subscriber's schema.

Syntax

```
DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW (
   subscription_handle  IN NUMBER,
   source_schema        IN VARCHAR2,
   source_table         IN VARCHAR2);
```

Parameters

Table 27–12 DROP_SUBSCRIBER_VIEW Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.
source_schema	Schema name where the source table resides.
source_table	Name of the published source table that belongs to the subscription handle.

Exceptions

Table 27–13 DROP_SUBSCRIBER_VIEW Procedure Exceptions

Exception	Description
ORA-31425	Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.
ORA-31429	The subscription has not been activated. Check the subscription handle and correct it, if necessary. Call the ACTIVATE_SUBSCRIPTION procedure for this subscription handle and then try the original command again.

Table 27–13 DROP_SUBSCRIBER_VIEW Procedure Exceptions (Cont.)

Exception	Description
ORA-31432	The schema_name.source_table does not exist or does not belong to this subscription. Check the spelling of the schema_name and source_table parameters. Verify the specified table exists in the specified schema and is subscribed to by the subscription handle.
ORA-31433	The subscriber view does not exist. Either you specified an incorrect source table or its view is already dropped.

Usage Notes

- The subscriber view you want to drop must have been created with a prior call to the DBMS_LOGMNR_CDC_SUBSCRIBE.PREPARE_SUBSCRIBER_VIEW procedure.
- You must use this procedure to drop the subscriber view prior to dropping a subscription using the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION procedure.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW( \
   SUBSCRIPTION_HANDLE =>:subhandle, \
   SOURCE_SCHEMA =>'scott', \
   SOURCE_TABLE => 'emp');
```

PURGE_WINDOW Procedure

The subscriber calls this procedure to notify the capture system it is finished processing a block of changes. The PURGE_WINDOW procedure sets the low-water mark so that the subscription no longer sees any data, effectively making the subscription window empty.

Syntax

```
DBMS_CDC_SUBSCRIBE.PURGE_WINDOW(
   subscription_handle   IN NUMBER);
```

Parameters

Table 27–14 PURGE_WINDOW Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.

Exceptions

Table 27–15 PURGE_WINDOW Procedure Exceptions

Exception	Description
ORA-31425	Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.
ORA-31429	The subscription handle must be activated before you use the EXTEND_WINDOW procedure. Call the ACTIVATE_SUBSCRIPTION procedure for this subscription handle and then try the original command again.
ORA-31430	The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW Procedure and then try the original command again.

Usage Notes

- When finished with a set of changes, the subscriber purges the subscription window with the PURGE_WINDOW procedure. By this action the subscriber performs the following functions:
  - Informs the change capture system that the subscriber is ready to receive the next batch of change data.
  - Enables the system to remove change data that is no longer needed by any subscribers.

  The Change Data Capture system manages the change data to ensure that it is available as long as there are subscribers who need it.

Example

```
EXECUTE sys.DBMS_CDC_SUBSCRIBE.PURGE_WINDOW ( \
SUBSCRIPTION_HANDLE=>:subhandle);
```

DROP_SUBSCRIPTION Procedure

This procedure drops a subscription that was created with a prior call to the GET_SUBSCRIPTION_HANDLE procedure.

Syntax

```
DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION (
   subscription_handle  IN NUMBER);
```

Parameters

Table 27–16 DROP_SUBSCRIPTION Procedure Parameters

Parameter	Description
subscription_handle	Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.

Exceptions

Table 27–17 DROP_SUBSCRIPTION Procedure Exceptions

Exception	Description
ORA-31425	Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.
ORA-31430	The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW procedure and then try the original command again.

Usage Notes

- Prior to dropping a subscription, you must drop the subscriber view using the DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW procedure.

Example

```
EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION (\
   SUBSCRIPTION_HANDLE => :subhandle);
```

28

DBMS_LOGMNR_D

The DBMS_LOGMNR_D package contains the LogMiner procedures, DBMS_LOGMNR_D.BUILD and DBMS_LOGMNR_D.SET_TABLESPACE. The DBMS_LOGMNR_D.BUILD procedure extracts the dictionary to either the redo logs or to a flat file. This information is saved in preparation for future analysis of redo logs using the LogMiner tool. The DBMS_LOGMNR_D.SET_TABLESPACE procedure re-creates all LogMiner tables in an alternate tablespace.

> **See Also:** *Oracle9i Database Administrator's Guide* for information about using LogMiner

This chapter discusses the following topics:

Summary of DBMS_LOGMNR_D Subprograms

Table 28–1 describes the procedures in the DBMS_LOGMNR_D supplied package.

Table 28–1 DBMS_LOGMNR_D Package Subprograms

Subprogram	Description
BUILD Procedure on page 28-2	Extracts the database dictionary to either a flat file or a file in the redo logs.
SET_TABLESPACE Procedure on page 28-5	Re-creates all LogMiner tables in an alternate tablespace.

BUILD Procedure

The syntax for the DBMS_LOGMNR_D.BUILD procedure is as follows:

Syntax

```
DBMS_LOGMNR_D.BUILD (
dictionary_filename IN VARCHAR2,
dictionary_location IN VARCHAR2,
options IN NUMBER);
```

Parameters

Table 28–2 describes the parameters for the BUILD procedure.

Table 28–2 BUILD Procedure Parameters

Parameter	Description
dictionary_filename	Name of the dictionary file
dictionary_location	Path to file directory
options	Specifies that the dictionary is written to either a flat file (STORE_IN_FLAT_FILE) or the redo logs (STORE_IN_REDO_LOGS) destination

To extract the dictionary to a flat file, you must supply a filename and location.

To extract the dictionary to the redo logs, specify only the STORE_IN_REDO_LOGS option. The size of the dictionary may cause it to be contained in multiple redo logs.

In summary, the combinations of parameters used result in the following behavior:

- If you do not specify any parameters, an error message is returned.

- If you specify a filename and location, without any options, the dictionary is extracted to a flat file with that name.
- If you specify a filename and location, as well as the DBMS_LOGMNR_D.STORE_IN_FLAT_FILE option, the dictionary is extracted to a flat file with the specified name.
- If you do not specify a filename and location, but do specify the DBMS_LOGMNR_D.STORE_IN_REDO_LOGS option, the dictionary is extracted to the redo logs.
- If you specify a filename and location, as well as the STORE_IN_REDO_LOGS option, an error is returned.

Exceptions

- ORA-1308: initialization parameter UTL_FILE_DIR is not set.
- ORA-1336 - this error is returned under the following conditions:
  1. Dictionary_location does not exist.
  2. UTL_FILE_DIR is not set to have access to dictionary_location.
  3. Dictionary file is read only.

Usage Notes

- Ideally, the dictionary file will be created after all dictionary changes to a database and prior to the creation of any redo logs that are to be analyzed. As of Oracle9*i* release 1 (9.0.1), you can use LogMiner to dump the dictionary to the redo logs, perform DDL operations, and dynamically apply the changes to the LogMiner dictionary.
- The DBMS_LOGMNR_D.BUILD procedure will not run if there are any ongoing DDL operations.
- To use the DBMS_LOGMNR_D.BUILD procedure, the database whose files you want to analyze must be mounted and open.
- To monitor progress of the dictionary build, issue the SET SERVEROUTPUT ON command.
- When extracting a dictionary to a flat file, the procedure queries the dictionary tables of the current database and creates a text-based file containing the contents of the tables. To extract a dictionary to a flat file, the following conditions must be met:

  - The dictionary file must be created from the same database that generated the redo logs you want to analyze
  - You must specify a directory for use by the PL/SQL procedure. To do so, set the initialization parameter `UTL_FILE_DIR` in the `init.ora` file. For example:

    ```
    UTL_FILE_DIR = /oracle/dictionary
    ```

    If you do not set this parameter, the procedure will fail.
  - You must ensure that no DDL operations occur while the dictionary build is running. Otherwise, the dictionary file may not contain a consistent snapshot of the data dictionary.

- To extract a dictionary file to the redo logs, the following conditions must be met:
  - Supplemental logging (at least the minimum level) must be enabled to ensure that the redo logs contain useful information. See *Oracle9i Database Administrator's Guide* for information about using supplemental logging with LogMiner.
  - The `DBMS_LOGMNR_D.BUILD` procedure must be run on a system that is running Oracle9*i* or later
  - Archiving mode must be enabled in order to generate usable redo
  - Oracle9*i* compatibility must be employed
  - The mining system must be Oracle9*i* or later
  - The dictionary redo files must be created from the same database that generated the redo logs you want to analyze

Example 1: Extracting the Dictionary to a Flat File

The following example extracts the dictionary file to a flat file named `dictionary.ora` in a specified path (`/oracle/database`).

```
SQL> EXECUTE dbms_logmnr_d.build('dictionary.ora', -
  2 '/oracle/database/', -
  3 options => dbms_logmnr_d.store_in_flat_file);
```

Example 2: Extracting the Dictionary to the Redo Logs

The following example extracts the dictionary to the redo logs.

```
SQL> EXECUTE dbms_logmnr_d.build ( -
```

```
2 options => dbms_logmnr_d.store_in_redo_logs);
```

SET_TABLESPACE Procedure

By default all LogMiner tables are created to use the SYSTEM tablespace. However, it may be desirable to alter LogMiner tables to employ an alternate tablespace. Use this routine to re-create all LogMiner tables in an alternate tablespace.

Parameters

Table 28–3 describes the parameters for the SET_TABLESPACE procedure.

Table 28–3 SET_TABLESPACE Parameters

Parameter	Description
new_tablespace	A string naming a preexistent tablespace. To re-create all LogMiner tables to employ this tablespace, supply only this parameter.
dictionary_ tablespace	A string naming a preexistent tablespace. This parameter places LogMiner Dictionary data in a tablespace different from that where LogMiner spill data is to be written. This parameter overrides the new_tablespace parameter with respect to LogMiner Dictionary tables.
spill_tablespace	A string naming a preexistent tablespace. This parameter places LogMiner spill data in a tablespace different from that where LogMiner Dictionary data is to be written. This parameter overrides the new_tablespace parameter with respect to LogMiner spill tables.

Usage Notes

- There can be no LogMiner sessions running at the time this procedure is run, nor can LogMiner have been terminated abnormally prior to this procedure being run. Either situation can cause unpredictable results.
- Though the intent is that this routine is to be run only once to configure LogMiner for use by other products, it can be run multiple times should it be necessary to redefine the tablespaces that are to be employed. However, the previous usage note is still enforced. Because the techniques required to force layered products to terminate their LogMiner sessions may be non-trivial, Oracle Corporation does not recommend that this routine be used more than once.

- Certain layered products require that this routine be used to alter the tablespace of all LogMiner tables before the layered product will operate.
- Certain performance optimizations can be made when LogMiner tables do not employ the SYSTEM tablespace. Specifically, certain easily repeatable operations, such as memory spill, LogMiner dictionary load, and index creation will not be logged. This would have unacceptable implications with respect to the SYSTEM tablespace in the event of a database recovery.
- Users of this routine must supply an existing tablespace. Information about tablespaces and how to create them is available in *Oracle9i Database Concepts* and *Oracle9i SQL Reference*.

Example: Using the DBMS_LOGMNR_D.SET_TABLESPACE Procedure

The following example shows creation of an alternate tablespace and execution of the DBMS_LOGMNR_D.SET_TABLESPACE procedure.

```
SQL> CREATE TABLESPACE  logmnrts$ datafile '/usr/oracle/dbs/logmnrts'
  2 SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

SQL> EXECUTE dbms_logmnr_d.set_tablespace('logmnrts$');
```

29

DBMS_LOGSTDBY

The DBMS_LOGSTDBY package provides procedures for configuring and managing the logical standby database environment.

> **See Also:** *Oracle9i Data Guard Concepts and Administration* for more information about logical standby databases.

This chapter discusses the following topics:

- Configuring and Managing the Logical Standby Environment
- Summary of DBMS_LOGSTDBY Subprograms

Configuring and Managing the Logical Standby Environment

The DBMS_LOGSTDBY package provides procedures to help you manage the logical standby environment. The procedures in the DBMS_LOGSTDBY package help you to accomplish the following main objectives:

- Allow controlled access to tables in the standby database that may require maintenance
- Provide a way to skip applying archived redo logs to selected tables or entire schemas in the standby database, and describe how exceptions should be handled
- Manage initialization parameters used by log apply services
- Ensure supplemental logging is enabled properly and build the LogMiner dictionary

Summary of DBMS_LOGSTDBY Subprograms

Table 29–1 describes the procedures of the DBMS_LOGSTDBY package.

Table 29–1 DBMS_LOGSTDBY Package Subprograms

Subprograms	Description
APPLY_SET Procedure on page 29-3	Allows you to set the values of specific initialization parameters to configure and maintain log apply services
APPLY_UNSET Procedure on page 29-7	Resets the value of specific initialization parameters to the system default values.
BUILD Procedure on page 29-8	Ensures supplemental logging is enabled properly and builds the LogMiner dictionary
GUARD_BYPASS_OFF Procedure on page 29-9	Re-enables the database guard that you bypassed previously with the GUARD_BYPASS_ON procedure
GUARD_BYPASS_ON Procedure on page 29-9	Allows the current session to bypass the database guard so that tables in a logical standby database can be modified
INSTANTIATE_TABLE Procedure on page 29-10	Creates and populates a table in the standby database from a corresponding table in the primary database

Table 29–1 (Cont.) DBMS_LOGSTDBY Package Subprograms

Subprograms	Description
SKIP Procedure on page 29-11	Allows you to specify what database operations that are done on the primary database will *not* be applied to the logical standby database
SKIP_ERROR Procedure on page 29-18	Specifies criteria to follow if an error is encountered. You can choose to stop log apply services or ignore the error
SKIP_TRANSACTION Procedure on page 29-21	Specifyies transaction identification information to skip (ignore) applying specific transactions to the logical standby database
UNSKIP Procedure on page 29-22	Modifies the options set in the SKIP procedure
UNSKIP_ERROR Procedure on page 29-23	Modifies the options set in the SKIP_ERROR procedure
UNSKIP_TRANSACTION Procedure on page 29-23	Modifies the options set in the SKIP_TRANSACTION procedure

APPLY_SET Procedure

Use this procedure to set and modify the values of initialization parameters that configure and manage log apply services in a logical standby database environment. Log apply services cannot be running when you use this procedure.

Syntax

```
DBMS_LOGSTDBY.APPLY_SET (
     parameter          IN VARCHAR,
     value              IN VARCHAR);
```

Parameters

Table 29–2 describes the parameters for the APPLY_SET procedure.

Table 29–2 DBMS_LOGSTDBY.APPLY_SET Procedure Parameters

Parameter	Description
APPLY_DELAY	Specifies an apply delay interval (in minutes) to the managed recovery operation on the standby database. Use the APPLY_DELAY parameter with the APPLY_UNSET procedure after a failover scenario, when the primary database is not expected to return.

Table 29–2 (Cont.) DBMS_LOGSTDBY.APPLY_SET Procedure Parameters

Parameter	Description
MAX_SGA	Number of megabytes for the system global area (SGA) allocation for log apply services cache. The default value is one quarter of the value set for the SHARED_POOL_SIZE initialization parameter.
MAX_SERVERS	Number of parallel query servers specifically reserved for log apply services. By default, log apply services use all available parallel query servers to read the log files and apply changes. See *Oracle9i Database Reference* for more information about parallel query servers.
MAX_EVENTS_RECORDED	Number of events that will be stored in the DBA_LOGSTDBY_EVENTS table, which stores logical standby event information.

Table 29–2 (Cont.) DBMS_LOGSTDBY.APPLY_SET Procedure Parameters

Parameter	Description
TRANSACTION_CONSISTENCY	Level of transaction consistency maintained between the primary and standby databases. Specify one of the following values: FULL: Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. (Transactions are applied in commit SCN order.) This option results in the lowest performance. This is the default parameter setting. NONE: Transactions are applied out of order. This results in the best performance of the three modes. However, this setting might give you inconsistent results on the standby database. If applications that are reading the logical standby database make no assumptions about transaction order, this option works well. For example, on the primary database, one transaction added a new customer and a second transaction added a new order for that customer. On the standby database, those transactions may be reversed. The order for the new customer might be added first. If you then run a reporting application on the standby database which expects to find a customer for the new order, the reporting application might fail because constraints are not checked and triggers are not fired. READ_ONLY: Transactions are committed out of order (which provides better performance), but periodically enforced in order apply. SQL SELECT statements, executed on the standby database, always return consistent results based on the last consistent SCN known to the apply engine. The apply engine periodically refreshes an SCN maintained in SGA which represents a consistent state. Queries executed on the standby database, automatically use Oracle Flashback to the maintained SCN. This is beneficial when the logical standby database is being used to generate reports. Any Oracle Flashback restrictions apply to this mode.
RECORD_SKIP_ERRORS	Controls whether skipped errors (as described by the SKIP_ERROR procedure) are recorded in the DBA_LOGSTDBY_EVENTS table. Specify one of the following values: TRUE: Skipped errors are recorded in the DBA_LOGSTDBY_EVENTS table. This is the default parameter setting. FALSE: Skipped errors are not recorded in the DBA_LOGSTDBY_EVENTS table.

Table 29–2 (Cont.) DBMS_LOGSTDBY.APPLY_SET Procedure Parameters

Parameter	Description
RECORD_SKIP_DDL	Controls whether skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table. Specify one of the following values: TRUE: Skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table. This is the default parameter setting. FALSE: Skipped DDL statements are not recorded in the DBA_LOGSTDBY_EVENTS table.
RECORD_APPLIED_DDL	Controls whether DDL statements that have been applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table. Specify one of the following values: TRUE: Indicates that DDL statements applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table. This is the default parameter setting. FALSE: Indicates that applied DDL statements are not recorded.

Exceptions

Table 29–3 describes the exceptions for the APPLY_SET procedure.

Table 29–3 DBMS_LOGSTDBY.APPLY_SET Procedure Exceptions

Exception	Description
ORA-16104	Invalid option
ORA-16103	Logical standby database must be stopped

Usage Notes

- Although the default values provided by the system for initialization parameters are adequate for most applications, you might want to use the APPLY_SET procedure when you need to perform tuning and maintenance tasks. For example, use the APPLY_SET procedure when you want to override default initialization parameter values to tune log apply services.
- Log apply services must not be applying archived redo log data to the standby database when you modify initialization parameters with the APPLY_SET

procedure. The initialization parameter values that you set using this procedure do not become active until you start log apply services.

- When a primary database is no longer available after failover, use the DBMS_LOGSTDBY.APPLY_UNSET('APPLY_DELAY') procedure to remove the setting provided by the initialization parameter file.
- Use the APPLY_UNSET Procedure to reverse (undo) the actions of the APPLY_SET procedure.

Example

If parallel queries are routinely being performed by applications, a certain number of parallel servers should be reserved for those queries. To allocate 30 parallel query servers for logical standby log apply services, enter the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_SERVERS', 30);
```

Thus, if the PARALLEL_MAX_SERVERS parameter is set to 50, 30 servers will be available for logical standby processing and 20 parallel query servers will be allocated for parallel query processing.

> **Note:** If you start log apply services while a parallel query is running, you may get an error.

APPLY_UNSET Procedure

Use the APPLY_UNSET procedure to reverse or undo the settings that you made with the APPLY_SET procedure. The APPLY_UNSET procedure resets the specified initialization parameter value to the system default value. The initialization parameter default value does not become active until log apply services are started.

Syntax

```
DBMS_LOGSTDBY.APPLY_UNSET (
     parameter          IN VARCHAR);
```

Parameters

The APPLY_UNSET procedure supports the same initialization parameters shown for the APPLY_SET procedure.

> **See Also:** Table 29–2 for the APPLY_SET procedure parameters

Usage Notes

- Log apply services must not be applying archived redo log data to the standby database when you modify initialization parameters with the APPLY_UNSET procedure.
- Use the APPLY_SET procedure to set the values of initialization parameters.

Example

To unset the number of parallel query servers for log apply services, enter the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('MAX_SERVERS');
```

Assuming that the PARALLEL_MAX_SERVERS initialization parameter is set to 50, this statement will result in 50 parallel query servers being available for parallel query processing. This is because, by default, log apply services use all available parallel query servers to read the log files and apply changes.

> **Note:** If you start log apply services while a parallel query is running, you may get an error.

BUILD Procedure

Use this procedure on the primary database to preserve important metadata (LogMiner dictionary) information in the redo logs. If supplemental logging has not been set correctly, this procedure sets it up and enables it automatically.

Syntax

```
DBMS_LOGSTDBY.BUILD;
```

Parameters

None.

Exceptions

None.

Usage Notes

- Supplemental log information includes extra information in the archived redo logs that helps log apply services to uniquely identify and correctly maintain tables in a logical standby database.

- LogMiner dictionary information allows log apply services to interpret data in the redo logs.

GUARD_BYPASS_OFF Procedure

Use the GUARD_BYPASS_OFF procedure to re-enable the database guard that you bypassed previously with the GUARD_BYPASS_ON Procedure procedure.

Syntax

```
DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
```

Parameters

None.

Exceptions

None.

Usage Notes

- See the GUARD_BYPASS_ON Procedure procedure for information about bypassing the database guard and performing maintenance on a table in the logical standby database.

Example

Enter the following statement to return the current session to the state it was in before the GUARD_BYPASS_ON Procedure was executed.

```
SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
```

Typically, you need to use this command only after executing the GUARD_BYPASS_ON Procedure to bypass the database guard.

GUARD_BYPASS_ON Procedure

By default, tables in a logical standby database are protected from modifications. However, you can use the GUARD_BYPASS_ON procedure to bypass the database guard and make modifications to the logical standby database. For example, to perform maintenance or correct problems on a table in the logical standby database. Applications should not execute transactions against the database when you use this procedure, because triggers are not run and constraints are not checked.

Syntax

```
DBMS_LOGSTDBY.GUARD_BYPASS_ON;
```

Parameters

None.

Exceptions

None.

Usage Notes

- This procedure affects the current session only.
- When you bypass the database guard with the GUARD_BYPASS_ON procedure, triggers are not run and constraints are not checked.
- Do not allow applications to execute when the use the GUARD_BYPASS_ON procedure to bypass the database guard. This environment is intended only for maintenance reasons, such as to correct problems or to perform maintenance such as rebuilding indexes or refreshing materialized views.

Example

Enter the following statement to allow modifications to tables in the logical standby database.

```
SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
```

INSTANTIATE_TABLE Procedure

This procedure creates and populates a table in the standby database from a corresponding table in the primary database. The table requires the name of the database link (dblink) as an input parameter.

Use the INSTANTIATE_TABLE procedure to:

- Add a table to a standby database
- Re-create a table in a standby database

Syntax

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
     table_name          IN VARCHAR2,
```

```
schema_name          IN VARCHAR2,
dblink               IN VARCHAR2);
```

Parameters

Table 29–4 describes the parameters for the INSTANTIATE_TABLE procedure.

Table 29–4 DBMS_LOGSTDBY.INSTANTIATE_TABLE Procedure Parameters

Parameter	Description
table_name	Name of the table to be created or re-created in the standby database.
schema_name	Name of the schema.
dblink	Name of the database link account that has privileges to read and lock the table in the primary database.

Exceptions

None.

Usage Notes

- Use this procedure to create and populate a table in a way that keeps the data on the standby database transactionally consistent with the primary database.
- This procedure assumes that the metadata has been maintained correctly.
- This table is not safe until the redo log that was current on the primary database at the time of execution is applied to the standby database.

Example

Enter this statement to create and populate a new table on the standby database.

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE ('myschema', 'mytable', 'mydblink');
```

SKIP Procedure

By default, all SQL statements executed on a primary database are applied to a logical standby database. If only a subset of activity on a primary database is of interest for replication, the SKIP procedure defines filters that prevent the application of SQL statements on the logical standby database. While skipping (ignoring) SQL statements is the primary goal of filters, it is also possible to associate a stored procedure with a filter so that runtime determinations can be

made whether to skip the statement, execute this statement, or execute a replacement statement.

Before calling this procedure, log apply services must be halted. This is done by issuing an ALTER DATABASE STOP LOGICAL STANDBY APPLY statement. Once all desired filters have been specified, issue an ALTER DATABASE START LOGICAL STANDBY APPLY statement to start log apply services using the new filter settings.

Syntax

```
DBMS_LOGSTDBY.SKIP (
     statement_option          IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2,
     proc_name                 IN VARCHAR2);
```

Parameters

Table 29–5 describes the parameters for the SKIP procedure.

Table 29–5 DBMS_LOGSTDBY.SKIP Procedure Parameters

Parameter	Description
statement_option	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 29–6 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.
schema_name	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the statement_option parameter. If not applicable, this value must be set to NULL.
object_name	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the statement_option. If not applicable, this value must be set to NULL.

Table 29–5 (Cont.) DBMS_LOGSTDBY.SKIP Procedure Parameters

Parameter	Description
proc_name	Name of a stored procedure to call when log apply services determines that a particular statement matches the filter defined by the statement_option, schema_name, and object_name parameters. Specify the procedure in the following format: '"schema"."package"."procedure"' This procedure returns a value that directs log apply services to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. Log apply services calls the stored procedure with the following call signature: ▪ IN STATEMENT VARCHAR2 -- The SQL statement that matches the filter ▪ IN STATEMENT_TYPE VARCHAR2 -- The statement_option of the filter ▪ IN SCHEMA VARCHAR2 -- The schema_name of the filter, if applicable ▪ IN NAME VARCHAR2 -- The object_name of the filter, if applicable ▪ IN XIDUSN NUMBER -- Transaction ID part 1 ▪ IN XIDSLT NUMBER -- Transaction ID part 2 ▪ IN XIDSQN NUMBER -- Transaction ID part 3 ▪ OUT SKIP_ACTION NUMBER -- Action to be taken by log apply services upon completion of this routine. Valid values are: SKIP_ACTION_APPLY -- Execute the statement SKIP_ACTION_SKIP -- Skip the statement SKIP_ACTION_REPLACE -- Execute the replacement statement supplied in the NEW_STATEMENT output parameter ▪ OUT NEW_STATEMENT VARCHAR2 -- The statement to execute in place of the original statement. Use of this option requires that SKIP_ACTION be set to SKIP_ACTION_REPLACE. Otherwise, set this option to NULL.

Caution: Atomic execution cannot be guaranteed if hardware or software failures stop log apply services. In a failure situation, a statement maybe executed more than once.

These stored procedures are not supported with DBMS_LOGSTDBY.SKIP('DML'...). If multiple wildcards match a given database statement object defined by the statement_option parameter, only one of the matching stored procedures will be called (alphabetically, by procedure).

Skip Statement Options

Table 29–6 lists the supported values for the statement_option parameter of the SKIP procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. Any of the SQL statements in the right column, however, are also valid values. Note that keywords are generally defined by database object.

Table 29–6 Supported Values for statement_option Parameter

Keyword	Associated SQL Statements
NON_SCHEMA_DDL	*All DDL that does not pertain to a particular schema*
SCHEMA_DLL	*All DDL that pertains to a particular schema*
DML	*Sequence operations such as* sequence.nextval
CLUSTER	CREATE CLUSTER AUDIT CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK DROP DATABASE LINK
DIMENSION	CREATE DIMENSION ALTER DIMENSION DROP DIMENSION
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY

Table 29–6 (Cont.) Supported Values for statement_option Parameter

Keyword	Associated SQL Statements
INDEX	CREATE INDEX ALTER INDEX DROP INDEX
PROCEDURE[1]	CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PROCEDURE
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	CREATE ROLE ALTER ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE
SESSION	Logons
SYNONYM	CREATE SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT *SQL_statements* NOAUDIT *SQL_statements*
SYSTEM GRANT	GRANT *system_privileges_and_roles* REVOKE *system_privileges_and_roles*

Table 29–6 (Cont.) Supported Values for statement_option Parameter

Keyword	Associated SQL Statements
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE TRUNCATE TABLESPACE
TRIGGER	CREATE TRIGGER ALTER TRIGGER with ENABLE and DISABLE clauses DROP TRIGGER ALTER TABLE with ENABLE ALL TRIGGERS clause ALTER TABLE with DISABLE ALL TRIGGERS clause
TYPE	CREATE TYPE CREATE TYPE BODY ALTER TYPE DROP TYPE DROP TYPE BODY
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE VIEW DROP VIEW

[1] Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

Exceptions

Table 29–7 describes an exception for the SKIP procedure.

Table 29–7 DBMS_LOGSTDBY.SKIP Procedure Exceptions

Exception	Description
ORA-16203	"Unable to interpret skip procedure return values." Indicates that a SKIP procedure has either generated an exception or has returned ambiguous values. You can identify the offending procedure by examining the DBA_LOGSTDBY_EVENTS view.

Usage Notes

- Use the SKIP procedure with caution, particularly when skipping DDL statements. If a CREATE TABLE statement is skipped, for example, you must also specify other DDL statements that refer to that table in the SKIP procedure. Otherwise, the statements will fail and cause an exception. When this happens, log apply services stop running.
- See the UNSKIP Procedure for information about reversing (undoing) the settings of the SKIP procedure.

Example

The following example shows how to use the SKIP procedure to skip (ignore) a schema on the logical standby database.

Example 1 Skip a Schema

To skip changes for a given schema, you must prevent log apply services from creating new objects in the schema and from modifying existing objects in the schema. In addition, the tablespace that supports the schema must not change. The following example demonstrates this using the SKIP procedure in a situation where schema *smith* has some number of tables defined in tablespace *bones* that we wish to ignore.

```
BEGIN
DBMS_LOGSTDBY.SKIP('SCHEMA_DDL', 'SMITH', '%', null);
DBMS_LOGSTDBY.SKIP('DML', 'SMITH', '%', null);
DBMS_LOGSTDBY.SKIP('TABLESPACE', null, null, 'SMITH.PROTECT_BONES');

END;
```

In the previous example, wildcards were used for the object_name parameter to indicate that the filter applies to all objects. In the last call to the SKIP procedure, the PROTECT_BONES procedure was supplied so that TABLESPACE could prevent tablespace operations on BONES. The following example is the definition for the PROTECT_BONES procedure:

```
CREATE OR REPLACE PROCEDURE PROTECT_BONES (statement      IN  VARCHAR2,
                                           statement_type IN  VARCHAR2,
                                           schema         IN  VARCHAR2,
                                           name           IN  VARCHAR2,
                                           xidusn         IN  NUMBER,
                                           xidslt         IN  NUMBER,
                                           xidsqn         IN  NUMBER,
```

```
                                        skip_action    OUT NUMBER,
                                        new_statement  OUT VARCHAR2) AS
  BEGIN
     -- Init
     new_statement := NULL;

     -- Guaranteed to be either CREATE, DROP, or TRUNCATE TABLESPACE
     IF statement LIKE '%TABLESPACE BONES%'
     THEN
         -- Skip the statement
         skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;
     ELSE
         -- Apply the statement
         skip_action := DBMS_LOGSTDBY.SKIP_ACTION_APPLY;
     END IF;
  END protect_bones;
```

SKIP_ERROR Procedure

Upon encountering an error, the logical standby feature uses the criteria contained in this procedure to determine if the error should cause log apply services to stop. All errors to be skipped are stored in system tables that describe how exceptions should be handled.

Syntax

```
DBMS_LOGSTDBY.SKIP_ERROR (
     statement_option          IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2,
     proc_name                 IN VARCHAR2);
```

Parameters

Table 29–8 describes the parameters for the SKIP_ERROR procedure.

Table 29–8 DBMS_LOGSTDBY.SKIP_ERROR Procedure Parameters

Parameter	Description
`statement_option`	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 29–6 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.
`schema_name`	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the `statement_option` parameter. If not applicable, this value must be set to `NULL`.
`object_name`	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the `statement_option`. If not applicable, this value must be set to `NULL`.

Table 29–8 (Cont.) DBMS_LOGSTDBY.SKIP_ERROR Procedure Parameters

Parameter	Description
proc_name	Name of a stored procedure to call when log apply services determines a particular statement matches the filter defined by the statement_option, schema_name, and object_name parameters. Specify the procedure in the following format: 'schema.package.procedure' This procedure returns a value that directs log apply services to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. Log apply services calls the stored procedure with the following call signature: ▪ IN STATEMENT VARCHAR(4000) -- The first 4K of the statement ▪ IN STATEMENT_TYPE VARCHAR2 -- The statement_option of the filter ▪ IN SCHEMA VARCHAR2 -- The schema_name of the filter, if applicable ▪ IN NAME VARCHAR2 -- The object_name of the filter, if applicable ▪ IN XIDUSN NUMBER -- Transaction ID part 1 ▪ IN XIDSLT NUMBER -- Transaction ID part 2 ▪ IN XIDSQN NUMBER -- Transaction ID part 3 ▪ IN ERROR VARCHAR(4000) -- Text of error to be recorded (optional) ▪ OUT NEW_ERROR VARCHAR(4000) -- Null or modified error text

Exceptions

None.

Usage Notes

- A stored procedure provided to the SKIP_ERROR procedure is called when log apply services encounter an error that could shut down the application of redo logs to the standby database.

  Running this stored procedure affects the error being written in the STATUS column of the DBA_LOGSTDBY_EVENTS table. The STATUS_CODE column

remains unchanged. If the stored procedure is to have no effect, that is, apply will be stopped, then the NEW_ERROR is written to the events table. To truely have no effect, set NEW_ERROR to ERROR in the procedure.

If the stored procedure requires that a shutdown be avoided, then you must set NEW_ERROR to NULL.

Example

```
DBMS_LOGSTDBY.SKIP_ERROR('DDL', 'joe', 'apptemp', null);
```

SKIP_TRANSACTION Procedure

This procedure provides a way to skip (ignore) applying transactions to the logical standby database. You can skip specific transactions by specifying transaction identification information.

You may want to use the SKIP_TRANSACTION procedure to:

- Skip a transaction that has already failed and that might otherwise cause log apply services to stop.
- Skip a transaction that may logically corrupt data.

If log apply services stop due to a particular transaction (for example, a DDL transaction), you can specify that transaction ID and then continue to apply. You can call this procedure multiple times for as many transactions as you want log apply services to ignore.

> **Note:** Do not let the primary and logical standby databases diverge when skipping transactions. If possible, you should manually execute a compensating transaction in place of the skipped transaction.

Syntax

```
DBMS_LOGSTDBY.SKIP_TRANSACTION (
     XIDUSN NUMBER          STRING,
     XIDSLT NUMBER          STRING,
     XIDSQN NUMBER          STRING);
```

Parameters

Table 29–9 describes the parameters for the SKIP_TRANSACTION procedure.

Table 29–9 DBMS_LOGSTDBY.SKIP_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN NUMBER	Transaction ID undo segment number of the transaction being skipped.
XIDSLT NUMBER	Transaction ID slot number of the transaction being skipped.
XIDSQN NUMBER	Transaction ID sequence number of the transaction being skipped.

Usage Notes

- View the last statement in DBA_LOGSTDBY_EVENTS to determine the reason that log apply services stopped processing transactions to the logical standby database. Examine the statement and error condition provided.
- Use the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by log apply services.

Exceptions

None.

UNSKIP Procedure

This procedure reverses the actions of the SKIP procedure by finding the record, matching all the parameters, and removing the record from the system table. The match must be exact, and multiple *skip* actions can be undone only by a matching number of *unskip* actions. You cannot undo multiple skip actions using wildcard characters.

Syntax

```
DBMS_LOGSTDBY.UNSKIP (
     statement_option          IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2);
```

Parameters

The parameter information for the UNSKIP procedure is the same as that described for the SKIP procedure. See Table 29–5 for complete parameter information.

Exceptions

None.

UNSKIP_ERROR Procedure

This procedure reverses or undoes the actions of the SKIP_ERROR procedure by finding the record, matching all the parameters, and removing the record from the system table. The match must be exact, and multiple *skip* actions can be undone only by a matching number of *unskip* actions. You cannot undo multiple skip actions with just one unskip procedure call.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_ERROR (
     statement_option          IN VARCHAR2,
     schema_name               IN VARCHAR2,
     object_name               IN VARCHAR2);
```

Parameters

The parameter information for the UNSKIP_ERROR procedure is the same as that described for the SKIP_ERROR procedure. See Table 29–8 for complete parameter information.

Exceptions

None.

Example

```
DBMS_LOGSTDBY.UNSKIP_ERROR;
```

UNSKIP_TRANSACTION Procedure

This procedure reverses the actions of the SKIP_TRANSACTION procedure. The match must be exact, and multiple *skip transaction* actions can be undone only by a matching number of *unskip transaction* actions. You cannot undo multiple skip transaction actions using wildcard characters.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_TRANSACTION (
     XIDUSN NUMBER          STRING,
     XIDSLT NUMBER          STRING,
```

```
    XIDSQN NUMBER          STRING);
```

Parameters

Table 29–10 describes the parameters for the UNSKIP_TRANSACTION procedure.

Table 29–10 DBMS_LOGSTDBY.UNSKIP_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN NUMBER	Transaction ID undo segment number of the transaction being skipped.
XIDSLT NUMBER	Transaction ID slot number of the transaction being skipped.
XIDSQN NUMBER	Transaction ID sequence number of the transaction being skipped.

Usage Notes

- Use the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by log apply services.

Exceptions

None.

30

DBMS_METADATA

With DBMS_METADATA you can retrieve complete database object definitions (metadata) from the dictionary by specifying:

- The type of object, for example, tables, indexes, or procedures
- Optional selection criteria, such as owner or name
- Parse items (attributes of the returned objects that are to be parsed and returned separately).
- Optional transformations on the output. By default the output is represented in XML, but callers can specify transformations (into SQL DDL, for example), which are implemented by XSLT (Extensible Stylesheet Language Transformation) stylesheets stored in the database or externally.

DBMS_METADATA provides the following retrieval interfaces:

- For programmatic use: OPEN, SET_FILTER, SET_COUNT, GET_QUERY, SET_PARSE_ITEM, ADD_TRANSFORM, SET_TRANSFORM_PARAM, FETCH_xxx and CLOSE retrieve multiple objects.
- For use in SQL queries and for browsing: GET_XML and GET_DDL return metadata for a single named object. The GET_DEPENDENT_XML, GET_DEPENDENT_DDL, GET_GRANTED_XML, and GET_GRANTED_DDL interfaces return metadata for one or more dependent or granted objects.

This chapter discusses the following topics:

Summary of DBMS_METADATA Subprograms

Table 30–1 provides a summary of DBMS_METADATA subprograms.

Table 30–1 DBMS_METADATA Subprograms

Subprogram	Description
OPEN Procedure on page 30-2	Specifies the type of object to be retrieved, the version of its metadata, and the object model.
SET_FILTER Procedure on page 30-6	Specifies restrictions on the objects to be retrieved, for example, the object name or schema.
SET_COUNT Procedure on page 30-12	Specifies the maximum number of objects to be retrieved in a single FETCH_xxx call.
GET_QUERY Procedure on page 30-12	Returns the text of the queries that are used by FETCH_xxx.
SET_PARSE_ITEM Procedure on page 30-13	Enables output parsing by specifying an object attribute to be parsed and returned.
ADD_TRANSFORM Procedure on page 30-15	Specifies a transform that FETCH_xxx applies to the XML representation of the retrieved objects.
SET_TRANSFORM_PARAM Procedure on page 30-17	Specifies parameters to the XSLT stylesheet identified by transform_handle.
FETCH_xxx Procedure on page 30-21	Returns metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on.
CLOSE Procedure on page 30-24	Invalidates the handle returned by OPEN and cleans up the associated state.
GET_XML and GET_DDL Functions on page 30-28	Returns the metadata for the specified object as XML or DDL.
GET_DEPENDENT_XML and GET_DEPENDENT_DDL Functions on page 30-31	Returns the metadata for one or more dependent objects, specified as XML or DDL.
GET_GRANTED_XML and GET_GRANTED_DDL Functions on page 30-33	Returns the metadata for one or more granted objects, specified as XML or DDL.

OPEN Procedure

OPEN specifies the type of object to be retrieved, the version of its metadata, and the object model. The return value is an opaque context handle for the set of objects to be used in subsequent calls.

Syntax

```
DBMS_METADATA.OPEN (
   object_type  IN VARCHAR2,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'ORACLE', )
RETURN NUMBER;
```

Parameters

Table 30–2 provides descriptions of the parameters for the OPEN procedure.

Table 30–2 Open() Parameters

Parameter	Description
object_type	The type of object to be retrieved. Table 30–3 lists the valid type names and their meanings. These object types will be supported for the ORACLE model of metadata (see model in this table) in Oracle9*i*. Future models may support a different set of object types. The "Attributes" column specifies some object type attributes. Schema objects, such as tables, belong to schemas. Named objects have unique names (if they are schema objects, the name is unique to the schema). Dependent objects, such as indexes, are defined with reference to a base schema object. Granted objects are granted or assigned to a user or role and therefore have a named grantee. These differences are relevant when choosing object selection criteria. See "SET_FILTER Procedure" on page 30-6 for more information.
version	The version of metadata to be extracted. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are: COMPATIBLE (default)—the version of the metadata corresponds to the database compatibility level. Note that database compatibility must be set to 9.0.1 or higher. LATEST—the version of the metadata corresponds to the database version. A specific database version, for example, 9.0.1.
model	Specifies which view to use, because the API can support multiple views on the metadata. Only the ORACLE model is supported in Oracle9*i*.

Table 30–3 provides the name, meaning, attributes, and notes for the DBMS_METADATA package object types. In the attributes column, S represents a schema object, N represents a named object, D represents a dependent object, and G represents a granted object.

Table 30–3 DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
ASSOCIATION	associate statistics	D	
AUDIT	audits of SQL statements	DG	Modeled as dependent, granted object. The base object name is the statement audit option name (for example, ALTER SYSTEM). There is no base object schema. The grantee is the user or proxy whose statements are audited.
AUDIT_OBJ	audits of schema objects	D	None
CLUSTER	clusters	SN	None
COMMENT	comments	D	None
CONSTRAINT	constraints	SND	Does not include: ▪ primary key constraint for IOT ▪ column NOT NULL constraints ▪ certain REF SCOPE and WITH ROWID constraints for tables with REF columns
CONTEXT	application contexts	N	None
DB_LINK	database links	SN	Modeled as schema objects because they have owners. For public links, the owner is PUBLIC. For private links, the creator is the owner.
DEFAULT_ROLE	default roles	G	Granted to a user by ALTER USER
DIMENSION	dimensions	SN	None
DIRECTORY	directories	N	None
FUNCTION	stored functions	SN	None
INDEX	indexes	SND	None
INDEXTYPE	indextypes	SN	None

Table 30–3 (Cont.) DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
JAVA_SOURCE	Java sources	SN	None
LIBRARY	external procedure libraries	SN	None
MATERIALIZED_VIEW	materialized views	SN	None
MATERIALIZED_ VIEW_LOG	materialized view logs	D	None
OBJECT_GRANT	object grants	DG	None
OPERATOR	operators	SN	None
OUTLINE	stored outlines	N	None
PACKAGE	stored packages	SN	By default, both package specification and package body are retrieved. See "SET_FILTER Procedure" on page 30-6.
PACKAGE_SPEC	package specifications	SN	None
PACKAGE_BODY	package bodies	SN	None
PROCEDURE	stored procedures	SN	None
PROFILE	profiles	N	None
PROXY	proxy authentications	G	Granted to a user by ALTER USER
REF_CONSTRAINT	referential constraint	SND	None
ROLE	roles	N	None
ROLE_GRANT	role grants	G	None
ROLLBACK_SEGMENT	rollback segments	N	None
SEQUENCE	sequences	SN	None
SYNONYM	synonyms	See notes.	Private synonyms are schema objects. Public synonyms are not, but for the purposes of this API, their schema name is PUBLIC. The name of a synonym is considered to be the synonym itself. For example, in CREATE PUBLIC SYNONYM FOO FOR BAR, the resultant object is considered to have name FOO and schema PUBLIC.

Table 30–3 (Cont.) DBMS_METADATA: Object Types

Type Name	Meaning	Attributes	Notes
SYSTEM_GRANT	system privilege grants	G	None
TABLE	tables	SN	None
TABLESPACE	tablespaces	N	None
TABLESPACE_QUOTA	tablespace quotas	G	Granted with ALTER USER
TRIGGER	triggers	SND	None
TRUSTED_DB_LINK	trusted links	N	None
TYPE	user-defined types	SN	By default, both type and type body are retrieved. See "SET_FILTER Procedure" on page 30-6.
TYPE_SPEC	type specifications	SN	None
TYPE_BODY	type bodies	SN	None
USER	users	N	None
VIEW	views	SN	None
XMLSCHEMA	XML schema	SN	The object's name is its URL (which may be longer than 30 characters). Its schema is the user who registered it.

Returns

An opaque handle to the class of objects. This handle is used as input to SET_FILTER, SET_COUNT, ADD_TRANSFORM, GET_QUERY, SET_PARSE_ITEM, FETCH_xxx, and CLOSE.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OBJECT_PARAM. The version or model parameter was not valid for the object_type.

SET_FILTER Procedure

SET_FILTER specifies restrictions on the objects to be retrieved, for example, the object name or schema.

Syntax

```
DBMS_METADATA.SET_FILTER (
   handle  IN NUMBER,
   name    IN VARCHAR2,
   value   IN VARCHAR2);
DBMS_METADATA.SET_FILTER (
   handle  IN NUMBER,
   name    IN VARCHAR2,
   value   IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 30–4 describes the parameters for the SET_FILTER procedure.

Table 30–4 SET_FILTER Parameters

Parameter	Description
handle	The handle returned from OPEN.
name	The name of the filter. For each filter, Table 30–5 lists the object_type it applies to, its name, its datatype (text or Boolean) and its meaning or effect (including its default value, if any).
value	The value of the filter.

Table 30–5 describes the object type, name, datatype, and meaning of the filters available with the SET_FILTER procedure.

Table 30–5 SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
Named objects	NAME	text	Objects with this exact name are selected.
	NAME_EXPR	text	The filter value is the right-hand side of a SQL comparison, that is, a SQL comparison operator (=,!=, and so on) and the value compared against. The value must contain parentheses and quotation marks where appropriate. In PL/SQL and SQL*Plus, two single quotes (not a double quote) are needed to represent an apostrophe. For example: `'IN (''DEPT'',''EMP'')'` The filter value is combined with the object attribute corresponding to the object name to produce a WHERE condition in the query that fetches the objects. In the preceding example, objects named DEPT and EMP are retrieved. By default, all named objects of object_type are selected.
Schema objects	SCHEMA	text	Objects in this schema are selected.
	SCHEMA_EXPR	text	The filter value is the right-hand side of a SQL comparison. The filter value is combined with the object attribute corresponding to the object schema to produce a WHERE condition in the query that fetches the objects. See NAME_EXPR for syntax details. Default: - if BASE_OBJECT_SCHEMA is specified, then objects in that schema are selected; - otherwise, objects in the current schema are selected. See "Security" on page 30-10.
PACKAGE, TYPE	SPECIFICATION	Boolean	If TRUE, retrieve the package or type specification. Defaults to TRUE.
	BODY	Boolean	If TRUE, retrieve the package or type body. Defaults to TRUE.

Table 30–5 (Cont.) SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
TABLE	TABLESPACE	text	Objects in this tablespace (or having a partition in this tablespace) are selected.
	TABLESPACE_EXPR	text	The filter value is the right-hand side of a SQL comparison. The filter value is combined with the attribute corresponding to the object's tablespace (or in the case of a partitioned table, the partition's tablespaces) to produce a WHERE condition in the query that fetches the objects. See NAME_EXPR for syntax details. By default, objects in all tablespaces are selected.
Dependent Objects	BASE_OBJECT_NAME	text	Objects are selected that are defined or granted on objects with this name. Specify SCHEMA for triggers on schemas. Specify DATABASE for database triggers. Column-level comments cannot be selected by column name; the base object name must be the name of the table, view, or materialized view containing the column.
	BASE_OBJECT_SCHEMA	text	Objects are selected that are defined or granted on objects in this schema. If BASE_OBJECT_NAME is specified with a value other than SCHEMA or DATABASE, this defaults to the current schema.
INDEX, TRIGGER	SYSTEM_GENERATED	Boolean	If TRUE, select indexes or triggers even if they are system-generated. If FALSE, omit system-generated indexes or triggers. Defaults to TRUE.
Granted Objects	GRANTEE	text	Objects are selected that are granted to this user or role. Specify PUBLIC for grants to PUBLIC.
OBJECT_GRANT	GRANTOR	text	Object grants are selected that are granted by this user.

Table 30–5 (Cont.) SET_FILTER: Filters

Object Type	Name	Datatype	Meaning
SYNONYM, JAVA_ SOURCE, XMLSCHEMA	LONGNAME	text	A name longer than 30 characters. Objects with this exact name are selected. If the object name is 30 characters or less, the NAME filter must be used.
	LONGNAME_EXPR	text	The filter value is the right-hand side of a SQL comparison. The filter value is combined with the attribute corresponding to the object's long name to produce a WHERE condition in the query that fetches the objects. See NAME_EXPR for syntax details. By default, no filtering is done on the long name of an object.
All objects	CUSTOM_FILTER	text	The text of a WHERE condition. The condition is appended to the query that fetches the objects. By default, no custom filter is used. The other filters are intended to meet the needs of the majority of users. Use CUSTOM_FILTER when no defined filters exists for your purpose. Of necessity such a filter depends on the detailed structure of the UDTs and views used in the query that are defined in admin/catmeta.sql. Because filters may change from version to version, upward compatibility is not guaranteed.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_FILTER was called after the first call to FETCH_ xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_FILTER for the current OPEN context are permitted.
- INCONSISTENT_ARGS. The filter name is not valid for the object type associated with the OPEN context, or the filter value is the wrong datatype.

Security

With SET_FILTER, you can specify the schema of objects to be retrieved, but security considerations may override this specification. If the caller is SYS or has SELECT_CATALOG_ROLE, then any object can be retrieved; otherwise, only the following can be retrieved:

- Schema objects owned by the caller

- Public synonyms
- System privileges granted to the caller or to PUBLIC
- Grants on objects for which the caller is owner, grantor or grantee (either explicitly or as PUBLIC).

If you request objects that you are not privileged to retrieve, no exception is raised; the object is not retrieved, as if it did not exist.

Usage Notes

- You can use the same text expression filter multiple times with different values. All the filter conditions will be applied to the query. For example, to get objects with names between Felix and Oscar, do the following:

```
dbms_metadata.set_filter(handle,'NAME_EXPR','>=''FELIX''');
dbms_metadata.set_filter(handle,'NAME_EXPR','<=''OSCAR''');
```

- For dependent objects such as triggers, grants, and indexes, the following conditions apply:
  - When connected as a nonprivileged user — If BASE_OBJECT_NAME is specified as a filter, BASE_OBJECT_SCHEMA defaults to the current schema:

```
dbms_metadata.set_filter(h,'BASE_OBJECT_NAME','EMP');
```

  - When connected as a privileged user with SELECT_CATALOG_ROLE — The schema defaults to BASE_OBJECT_SCHEMA if specified; otherwise it defaults to the current schema. For example, to see all indexes in SCOTT that are defined on SCOTT.EMP, the filters are:

```
dbms_metadata.set_filter(h,'BASE_OBJECT_NAME','EMP');
dbms_metadata.set_filter(h,'BASE_OBJECT_SCHEMA','SCOTT');
```

    To see indexes in other schemas:

```
dbms_metadata.set_filter(h,'SCHEMA_EXPR','LIKE ''%''');
```

  Some indexes and triggers are system generated (such as indexes used to enforce unique constraints). Set the SYSTEM_GENERATED filter to FALSE so that you do not retrieve them.

SET_COUNT Procedure

SET_COUNT specifies the maximum number of objects to be retrieved in a single FETCH_xxx call. By default, each call to FETCH_xxx returns one object. With SET_COUNT, you can override this default. If FETCH_xxx is called from a client, specifying a count value greater than 1 can result in fewer server round trips and, therefore, improved performance. Note that the procedure stops when NULL is returned, but not if less than the maximum number of objects is returned.

Syntax

```
DBMS_METADATA.SET_COUNT (
   handle IN NUMBER,
   value  IN NUMBER);
```

Parameters

Table 30–6 describes the parameters for the SET_COUNT procedure.

Table 30–6 SET_COUNT Parameters

Parameter	Description
handle	The handle returned from OPEN.
value	The number of objects to retrieve.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_COUNT was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_COUNT for the current OPEN context are permitted.

GET_QUERY Procedure

GET_QUERY returns the text of the queries that are used by FETCH_xxx. This function assists in debugging.

Syntax

```
DBMS_METADATA.GET_QUERY (
   handle  IN NUMBER)
RETURN VARCHAR2;
```

Parameters

Table 30–7 describes the parameters for the GET_QUERY procedure.

Table 30–7 GET_QUERY Parameters

Parameter	Description
handle	The handle returned from OPEN.

Returns

The text of the queries that will be used by FETCH_xxx.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for the handle parameter.

SET_PARSE_ITEM Procedure

SET_PARSE_ITEM enables output parsing by specifying an object attribute to be parsed and returned. It should only be used in conjunction with FETCH_DDL.

Syntax

```
DBMS_METADATA.SET_PARSE_ITEM (
   handle   IN NUMBER,
   name     IN VARCHAR2);
```

Parameters

Table 30–8 describes the parameters for the SET_PARSE_ITEM procedure.

Table 30–8 SET_PARSE_ITEM Parameters

Parameter	Description
handle	The handle returned from OPEN.
name	The name of the object attribute to be parsed and returned. See Table 30–9 for the attribute object type, name, and meaning.

Table 30–9 describes the object type, name, and meaning of the items available in the SET_PARSE_ITEM procedure.

Table 30–9 SET_PARSE_ITEM: Parse Items

Object Type	Name	Meaning
All objects	VERB	For every row in the sys.ku$_ddls nested table returned by fetch_ddl the verb in the corresponding ddlText is returned. If the ddlText is a SQL DDL statement, then the SQL verb (for example, CREATE, GRANT, AUDIT) is returned. If the ddlText is a procedure call (for example., DBMS_RLS.ADD_POLICY_CONTEXT) then the package.procedure-name is returned.
	OBJECT_TYPE	If the ddlText is a SQL DDL statement whose verb is CREATE or ALTER, the object type as used in the DDL statement is returned, for example, TABLE, PACKAGE BODY, and so on. Otherwise, an object type name from Table 30–3, " DBMS_METADATA: Object Types" is returned.
	SCHEMA	The object schema is returned. If the object is not a schema object, NULL is returned.
	NAME	The object name is returned. If the object is not a named object, NULL is returned.
TABLE, INDEX	TABLESPACE	The tablespace name of the table or index is returned.
TRIGGER	ENABLE	If the trigger is enabled, ENABLE is returned. If the trigger is disabled, DISABLE is returned.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_PARSE_ITEM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_PARSE_ITEM are permitted.
- INCONSISTENT_ARGS. The attribute name is not valid for the object type associated with the OPEN context.

Usage Notes

By default fetch_ddl returns object metadata as creation DDL. By calling SET_PARSE_ITEM, you can request that individual attributes of the object be returned also, to avoid the tedious process of parsing SQL text. This is useful when fetching objects based on the value of a returned object, for example, fetching indexes for a returned table.

You can call SET_PARSE_ITEM multiple times to ask for multiple items to be parsed and returned. Parsed items are returned in the sys.ku$_parsed_items nested table. An example of using sys.ku$_parsed_items is shown within Example: Retrieving Payroll Tables and their Indexes as DDL on page 30-24.

See Also:

- "FETCH_xxx Procedure" on page 30-21
- *Oracle9i Database Utilities* for information about using the Metadata API

ADD_TRANSFORM Procedure

ADD_TRANSFORM specifies a transform that FETCH_xxx applies to the XML representation of the retrieved objects. It is possible to add more than one transform.

Syntax

```
DBMS_METADATA.ADD_TRANSFORM (
   handle    IN NUMBER,
   name      IN VARCHAR2,
   encoding  IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

Table 30–10 describes the parameters for the ADD_TRANSFORM procedure.

Table 30–10 ADD_TRANSFORM Parameters

Parameters	Description
handle	The handle returned from OPEN.

Table 30–10 (Cont.) ADD_TRANSFORM Parameters

Parameters	Description
name	The name of the transform. If the name is DDL, creation DDL will be generated using XSLT stylesheets stored within the Oracle dictionary. If the name contains a period (.), colon (:) or forward slash (/), it is interpreted as the URL of a user-supplied XSLT stylesheet. See *Oracle9i XML Database Developer's Guide - Oracle XML DB*.
encoding	The name of NLS character set (see National Language Support Guide) in which the stylesheet pointed to by name is encoded. This is only valid if the name is a URL. If left NULL and the URL is external to the database (e.g, /usr/williams/xsl/mystylesheet.xsl), UTF-8 encoding is assumed. If left NULL and the URL is internal to the database, that is, it begins with /oradb/, then the database character set is assumed to be the encoding.

Returns

An opaque handle to the transform. This handle is used as input to SET_TRANSFORM_PARAM. Note that this handle is different from the handle returned by OPEN; it refers to the transform, not the set of objects to be retrieved.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. ADD_TRANSFORM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to ADD_TRANSFORM for the current OPEN context are permitted.

Usage Notes

With no transforms added, objects are returned by default as XML documents. You call ADD_TRANSFORM to specify an XSLT stylesheet to transform the returned documents.

You can call ADD_TRANSFORM more than once to apply multiple transforms to the returned XML documents. FETCH_xxx will apply the transforms in the order in which they were specified, the output of the first transform being used as input to the second, and so on.

The encoding parameter must be specified if either of the following is true:

- The XSL stylesheet pointed to by an external URL is encoded in a character set that is not a subset of UTF-8
- The XSL stylesheet pointed to by a database-internal URL is encoded in a character set that is not a subset of the database character set.

An example of the latter might be if the database-internal URL pointed to an NCLOB or NVARCHAR column. Normally, this need not be specified, although explicitly setting it to US7ASCII (if applicable) results in slightly better XML parsing performance.

> **Note:** The output of the DDL transform is not an XML document. Therefore, no transform should be added after the DDL transform.

SET_TRANSFORM_PARAM Procedure

SET_TRANSFORM_PARAM specifies parameters to the XSLT stylesheet identified by transform_handle. Use it to modify or customize the output of the transform.

Syntax

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
   transform_handle  IN NUMBER,
   name                 IN VARCHAR2,
   value                IN VARCHAR2);
DBMS_METADATA.SET_TRANSFORM_PARAM (
   transform_handle  IN NUMBER,
   name                 IN VARCHAR2,
   value                IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 30–11 describes the parameters for the SET_TRANSFORM_PARAM procedure.

Table 30–11 SET_TRANSFORM_PARAM Parameters

Parameters	Description
transform_handle	Either (1) the handle returned from ADD_TRANSFORM, or (2) the enumerated constant SESSION_TRANSFORM that designates the DDL transform for the whole session. Note that the handle returned by OPEN is not a valid transform handle.

Table 30–11 (Cont.) SET_TRANSFORM_PARAM Parameters

Parameters	Description
name	The name of the parameter. Table 30–12 lists the transform parameters defined for the DDL transform, specifying the object_type it applies to, its datatype (in this case, always Boolean) and its meaning or effect (including its default value, if any).
value	The value of the transform.

Table 30–12 describes the object type, name, datatype, and meaning of the parameters for the DDL transform in the SET_TRANSFORM_PARAM procedure.

Table 30–12 SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
All objects	PRETTY	Boolean	If TRUE, format the output with indentation and line feeds. Defaults to TRUE.
	SQLTERMINATOR	Boolean	If TRUE, append a SQL terminator (; or /) to each DDL statement. Defaults to FALSE.
TABLE	SEGMENT_ATTRIBUTES	Boolean	If TRUE, emit segment attributes (physical attributes, storage attributes, tablespace, logging). Defaults to TRUE.
	STORAGE	Boolean	If TRUE, emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
	TABLESPACE	Boolean	If TRUE, emit tablespace. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.

Table 30–12 (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
TABLE	CONSTRAINTS	Boolean	If TRUE, emit all non-referential table constraints. Defaults to TRUE.
	REF_CONSTRAINTS	Boolean	If TRUE, emit all referential constraints (foreign key and scoped refs). Defaults to TRUE.
	CONSTRAINTS_AS_ALTER	Boolean	If TRUE, emit table constraints as separate ALTER TABLE (and, if necessary, CREATE INDEX) statements. If FALSE, specify table constraints as part of the CREATE TABLE statement. Defaults to FALSE. Requires that CONSTRAINTS be TRUE.
	OID	Boolean	If TRUE, emit the OID clause for object tables. Defaults to FALSE.
	SIZE_BYTE_KEYWORD	Boolean	If TRUE, emit the BYTE keyword as part of the size specification of CHAR and VARCHAR2 columns that use byte semantics. If FALSE, omit the keyword. Defaults to FALSE.
INDEX	SEGMENT_ATTRIBUTES	Boolean	If TRUE, emit segment attributes (physical attributes, storage attributes, tablespace, logging). Defaults to TRUE.
	STORAGE	Boolean	If TRUE, emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
	TABLESPACE	Boolean	If TRUE, emit tablespace. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
TYPE	SPECIFICATION	Boolean	If TRUE, emit the type specification. Defaults to TRUE.
	BODY	Boolean	If TRUE, emit the type body. Defaults to TRUE.
PACKAGE	SPECIFICATION	Boolean	If TRUE, emit the package specification. Defaults to TRUE.
	BODY	Boolean	If TRUE, emit the package body. Defaults to TRUE.

Table 30–12 (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

Object Type	Name	Datatype	Meaning
VIEW	FORCE	Boolean	If TRUE, use the FORCE keyword in the CREATE VIEW statement. Defaults to TRUE.
All objects	DEFAULT	Boolean	Calling SET_TRANSFORM_PARAM with this parameter set to TRUE has the effect of resetting all parameters for the transform to their default values. Setting this FALSE has no effect. There is no default.
	INHERIT	Boolean	If TRUE, inherits session-level parameters. Defaults to FALSE. If an application calls ADD_TRANSFORM to add the DDL transform, then by default the only transform parameters that apply are those explicitly set for that transform handle. This has no effect if the transform handle is the session transform handle.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OPERATION. SET_TRANSFORM_PARAM was called after the first call to FETCH_XXX for the OPEN context. After the first call to FETCH_XXX is made, no further calls to SET_TRANSFORM_PARAM are permitted.
- INCONSISTENT_ARGS. The transform parameter name is not valid for the object type associated with the OPEN context.

Usage Notes

XSLT allows parameters to be passed to stylesheets. You call SET_TRANSFORM_PARAM to specify the value of a parameter to be passed to the stylesheet identified by transform_handle. The most general way to specify stylesheet parameter values is as text strings. However, for the DDL transform, it is convenient to expose some parameters as Booleans. Consequently, two variants of the procedure are provided.

The GET_DDL function allows the casual browser to extract the creation DDL for an object. So that you can specify transform parameters, this package defines an enumerated constant SESSION_TRANSFORM as the handle of the DDL transform at the session level. You can call SET_TRANSFORM_PARAM using DBMS_METADATA.SESSION_TRANSFORM as the transform handle to set transform

parameters for the whole session. GET_DDL inherits these parameters when it invokes the DDL transform.

> **Note:** The enumerated constant must be prefixed with the package name DBMS_METADATA.SESSION_TRANSFORM.

FETCH_xxx Procedure

FETCH_xxx returns metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on. See "Usage Notes" on page 30-22 for the variants.

Syntax

The FETCH functions and procedures are:

```
DBMS_METADATA.FETCH_XML (
   handle  IN NUMBER)
RETURN sys.XMLType;
```

> **See Also:** *Oracle9i XML Database Developer's Guide - Oracle XML DB* for a description of XMLType

```
DBMS_METADATA.FETCH_DDL (
   handle  IN NUMBER)
RETURN sys.ku$_ddls;
```

The following types comprise the return nested table type sys.ku$_ddls:

```
TYPE sys.ku$_parsed_item AS OBJECT (
  item       VARCHAR2(30),
  value      VARCHAR2(4000),
  object-row NUMBER );
TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item;
TYPE sys.ku$_ddl AS OBJECT (
  ddlText CLOB,
  parsedItems sys.ku$_parsed_items );
TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl;

DBMS_METADATA.FETCH_CLOB (
   handle  IN NUMBER)
RETURN CLOB;
DBMS_METADATA.FETCH_CLOB (
   handle  IN NUMBER,
```

```
    doc      IN OUT NOCOPY CLOB);
```

Parameters

Table 30–13 describes the parameters for the FETCH_xxx procedure.

Table 30–13 FETCH_xxx Parameters

Parameters	Description
handle	The handle returned from OPEN.
doc (procedure fetch_clob)	The metadata for the objects or NULL if all objects have been returned.

Returns

The metadata for the objects or NULL if all objects have been returned.

Exceptions

Most exceptions raised during execution of the query are propagated to the caller. Also, the following exceptions may be raised:

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INCONSISTENT_OPERATION. Either (1) FETCH_XML was called when the DDL transform had been specified, or (2) FETCH_DDL was called when the DDL transform had not been specified.

Usage Notes

These functions and procedures return metadata for objects meeting the criteria established by calls to OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on. Each call to FETCH_xxx returns the number of objects specified by SET_COUNT (or less, if fewer objects remain in the underlying cursor) until all objects have been returned. After the last object is returned, subsequent calls to FETCH_xxx return NULL and cause the stream created by OPEN to be transparently closed.

There are several different FETCH_xxx functions and procedures:

- FETCH_XML returns the XML metadata for an object as an XMLType. It assumes that if any transform has been specified, the transform will produce an XML document. In particular, it assumes that the DDL transform has not been specified.

- FETCH_DDL returns the creation DDL in a sys.ku$_ddls nested table. It assumes that the DDL transform has been specified. Each row of the sys.ku$_ddls nested table contains a single DDL statement in the ddlText column; if requested, parsed items for the DDL statement will be returned in the parsedItems column. Multiple DDL statements may be returned under the following circumstances:
  - When you call SET_COUNT to specify a count greater than 1
  - When an object is transformed into multiple DDL statements. For example, A TYPE object can be transformed into both CREATE TYPE and CREATE TYPE BODY statements. A TABLE object can be transformed into a CREATE TABLE, zero or more CREATE INDEX statements, and zero or more ALTER TABLE statements.
- FETCH_CLOB simply returns the object, transformed or not, as a CLOB.

FETCH_CLOB comes in both function and procedure variants. The procedure variant returns the object by reference in an IN OUT NOCOPY parameter.

All LOBs returned by FETCH_xxx are temporary LOBs. You must free the LOB. The same applies to the XMLType object.

If SET_PARSE_ITEM was called, FETCH_DDL returns attributes of the DDL statement in a sys.ku$_parsed_items nested table, which is a column in the returned sys.ku$_ddls nested table. Each row of the sys.ku$_parsed_items nested table corresponds to an item specified by SET_PARSE_ITEM and contains the following columns:

- item—The name of the attribute as specified in the name parameter to SET_PARSE_ITEM.
- value—The attribute value, or NULL if the attribute is not present in the DDL statement.
- object-row—For future use.

The order of the rows is undetermined; to find a particular item you must search the table for a match on item.

If SET_PARSE_ITEM was not called, NULL is returned as the value of the sys.ku$_parsed_items nested table.

When Variants of FETCH_xxx Are Called

It is expected that the same variant of FETCH_xxx will be called for all objects selected by OPEN, that is, that programs will not intermix calls to FETCH_XML,

FETCH_DDL, and FETCH_CLOB using the same OPEN handle. The effect of calling different variants is undefined; it may not do what you expect.

CLOSE Procedure

CLOSE invalidates the handle returned by OPEN and cleans up the associated state.

Syntax

```
DBMS_METADATA.CLOSE (
   handle  IN NUMBER);
```

Parameters

Table 30–14 describes the parameters for the CLOSE procedure.

Table 30–14 CLOSE Parameters

Parameter	Description
handle	The handle returned from OPEN.

Exceptions

- INVALID_ARGVAL. The value for the handle parameter is NULL or invalid.

Usage Notes

You can prematurely terminate the stream of objects established by OPEN.

- If a call to FETCH_xxx returns NULL, indicating no more objects, a call to CLOSE is made transparently. In this case, you can still call CLOSE on the handle and not get an exception. (The call to CLOSE is not required.)
- If you know that only one specific object will be returned, you should explicitly call CLOSE after the single FETCH_xxx call to free resources held by the handle.

Example: Retrieving Payroll Tables and their Indexes as DDL

This example retrieves the creation DDL for all tables in the current schema whose names begin with PAYROLL. For each table it also returns the creation DDL for the indexes defined on the table. The returned DDL is written to an output file.

```
CREATE OR REPLACE PACKAGE dbms_metadata_example AS

   PROCEDURE get_payroll_tables;
```

```
END;
/
CREATE OR REPLACE PACKAGE BODY dbms_metadata_example AS

-- Global Variables

fileHandle   UTL_FILE.FILE_TYPE;

-- Exception initialization

file_not_found EXCEPTION;
PRAGMA EXCEPTION_INIT(file_not_found, -1309);

-- Package-private routine to write a CLOB to an output file.

  PROCEDURE write_lob(doc IN CLOB) IS

     outString     varchar2(32760);
     cloblen       number;
     offset        number := 1;
     amount        number;

BEGIN
  cloblen := dbms_lob.getlength(doc);
  WHILE cloblen > 0
  LOOP
    IF cloblen > 32760 THEN
      amount := 32760;
    ELSE
      amount := cloblen;
    END IF;
    outString := dbms_lob.substr(doc, amount, offset);
    utl_file.put(fileHandle, outString);
    utl_file.fflush(fileHandle);
    offset  := offset  + amount;
    cloblen := cloblen - amount;
  END LOOP;
  RETURN;
END;

-- Public routines

-- GET_PAYROLL_TABLES: Fetch DDL for payroll tables and their indexes.

PROCEDURE  get_payroll_tables IS
```

```
tableOpenHandle       NUMBER;
indexOpenHandle       NUMBER;
tableTransHandle      NUMBER;
indexTransHandle      NUMBER;
schemaName            VARCHAR2(30);
tableName             VARCHAR2(30);
tableDDLs             sys.ku$_ddls;
tableDDL              sys.ku$_ddl;
parsedItems           sys.ku$_parsed_items;
indexDDL              CLOB;

BEGIN

--  open the output file... note that the 1st param. (dir. path) must be
--  included in the database's UTL_FILE_DIR init. parameter.
--
  BEGIN
    fileHandle := utl_file.fopen('/private/xml', 'ddl.out', 'w', 32760);
  EXCEPTION
    WHEN OTHERS THEN
        RAISE file_not_found;
  END;

-- Open a handle for tables in the current schema.
  tableOpenHandle := dbms_metadata.open('TABLE');

-- Call 'set_count' to request retrieval of one table at a time.
-- This call is not actually necessary because 1 is the default.
  dbms_metadata.set_count(tableOpenHandle, 1);

-- Retrieve tables whose name starts with 'PAYROLL'. When the filter is
-- 'NAME_EXPR', the filter value string must include the SQL operator. This
-- gives the caller flexibility to use LIKE, IN, NOT IN, subqueries, and so on.
  dbms_metadata.set_filter(tableOpenHandle, 'NAME_EXPR', 'LIKE ''PAYROLL%''');

-- Tell Metadata API to parse out each table's schema and name separately
-- so we can use them to set up the calls to retrieve its indexes.
  dbms_metadata.set_parse_item(tableOpenHandle, 'SCHEMA');
  dbms_metadata.set_parse_item(tableOpenHandle, 'NAME');

-- Add the DDL transform so we get SQL creation DDL
  tableTransHandle := dbms_metadata.add_transform(tableOpenHandle, 'DDL');

-- Tell the XSL stylesheet we don't want physical storage information (storage,
```

```
-- tablespace, etc), and that we want a SQL terminator on each DDL. Notice that
-- these calls use the transform handle, not the open handle.
  dbms_metadata.set_transform_param(tableTransHandle,
      'SEGMENT_ATTRIBUTES', FALSE);
  dbms_metadata.set_transform_param(tableTransHandle,
      'SQLTERMINATOR', TRUE);

-- Ready to start fetching tables. We use the FETCH_DDL interface (rather than
-- FETCH_XML or FETCH_CLOB). This interface returns a SYS.KU$_DDLS; a table of
-- SYS.KU$_DDL objects. This is a table because some object types return
-- multiple DDL statements (like types / pkgs which have create header and
-- body statements). Each KU$_DDL has a CLOB containing the 'CREATE TABLE'
-- statement plus a nested table of the parse items specified. In our case,
-- we asked for two parse items; Schema and Name.

  LOOP
    tableDDLs := dbms_metadata.fetch_ddl(tableOpenHandle);
    EXIT WHEN tableDDLs IS NULL;   -- Get out when no more payroll tables

-- In our case, we know there is only one row in tableDDLs (a KU$_DDLS tbl obj)
-- for the current table. Sometimes tables have multiple DDL statements,
-- for example, if constraints are applied as ALTER TABLE statements,
-- but we didn't ask for that option.
-- So, rather than writing code to loop through tableDDLs,
-- we'll just work with the 1st row.
--
-- First, write the CREATE TABLE text to our output file, then retrieve the
-- parsed schema and table names.
    tableDDL := tableDDLs(1);
    write_lob(tableDDL.ddltext);
    parsedItems := tableDDL.parsedItems;

-- Must check the name of the returned parse items as ordering isn't guaranteed
    FOR i IN 1..2 LOOP
      IF parsedItems(i).item = 'SCHEMA'
      THEN
        schemaName := parsedItems(i).value;
      ELSE
        tableName  := parsedItems(i).value;
      END IF;
    END LOOP;

-- Then use the schema and table names to set up a 2nd stream for retrieval of
-- the current table's indexes.
-- (Note that we don't have to specify a SCHEMA filter for the indexes,
```

```
-- Because SCHEMA defaults to the value of BASE_OBJECT_SCHEMA.)
    indexOpenHandle := dbms_metadata.open('INDEX');
    dbms_metadata.set_filter(indexOpenHandle,'BASE_OBJECT_SCHEMA',schemaName);
    dbms_metadata.set_filter(indexOpenHandle,'BASE_OBJECT_NAME',tableName);

-- Add the DDL transform and set the same transform options we did for tables
    indexTransHandle := dbms_metadata.add_transform(indexOpenHandle, 'DDL');
    dbms_metadata.set_transform_param(indexTransHandle,
                      'SEGMENT_ATTRIBUTES', FALSE);
    dbms_metadata.set_transform_param(indexTransHandle,
                      'SQLTERMINATOR', TRUE);

-- Retrieve index DDLs as CLOBs and write them to the output file.
    LOOP
      indexDDL := dbms_metadata.fetch_clob(indexOpenHandle);
      EXIT WHEN indexDDL IS NULL;
      write_lob(indexDDL);
    END LOOP;

-- Free resources allocated for index stream.
    dbms_metadata.close(indexOpenHandle);

  END LOOP;

-- Free resources allocated for table stream and close output file.
  dbms_metadata.close(tableOpenHandle);
  utl_file.fclose(fileHandle);
  RETURN;

END;  -- of procedure get_payroll_tables

END dbms_metadata_example;
/
```

GET_XML and GET_DDL Functions

GET_XML and GET_DDL return the metadata for the specified object as XML or DDL.

Syntax

```
DBMS_METADATA.GET_XML (
   object_type  IN VARCHAR2,
   name         IN VARCHAR2,
   schema       IN VARCHAR2 DEFAULT NULL,
```

```
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'ORACLE',
   transform    IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;

DBMS_METADATA.GET_DDL (
   object_type  IN VARCHAR2,
   name         IN VARCHAR2,
   schema       IN VARCHAR2 DEFAULT NULL,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'ORACLE',
   transform    IN VARCHAR2 DEFAULT 'DDL')
RETURN CLOB;
```

Parameters

Table 30–15 describes the parameters for the GET_xxx function.

Table 30–15 GET_xxx Parameters

Parameter	Description
object_type	The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter.
name	An object name (case-sensitive). If object_type is SYNONYM and name is longer than 30 characters, then name will be treated as a LONGNAME filter. See Table 30–5.
schema	A schema name (case sensitive). The default is the current schema if object_type refers to a schema object; otherwise the default is NULL.
version	The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.
model	The object model to use. This parameter takes the same values as the OPEN model parameter.
transform	The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_XML this must not be DDL.

Returns

The metadata for the specified object as XML or DDL.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT_NOT_FOUND. The specified object was not found in the database.

Usage Notes

These functions provide a simple way to return the metadata for a single object. Conceptually each GET_xxx call is comprised of an OPEN, one or two SET_FILTER calls, optionally an ADD_TRANSFORM, a FETCH_xxx and a CLOSE. The object_type parameter has the same semantics as in OPEN. The schema and name parameters are used for filtering. If a transform is specified, schema-level transform flags are inherited.

This function can only be used to fetch named objects. It cannot be used to fetch objects of type OBJECT_GRANT or SYSTEM_GRANT. To fetch these objects, use the programmatic interface.

Example 1. Fetching the XML Representation of SCOTT.EMP

To generate complete, uninterrupted output, set the PAGESIZE to 0 and set LONG to some large number, as shown, before executing your query.

```
set pagesize 0
set long 90000
SELECT DBMS_METADATA.GET_XML
  (
   'TABLE','EMP','SCOTT')
    FROM DUAL;
```

Example 2. Fetching the DDL for all Complete Tables in the Current Schema, Filtering Out Nested Tables and Overflow Segments

This example fetches the DDL for all "complete" tables in the current schema, filtering out nested tables and overflow segments. The example uses SET_TRANSFORM_PARAM (with the handle value = DBMS_METADATA.SESSION_TRANSFORM meaning "for the current session") to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the example resets the session-level parameters to their defaults. (To generate complete, uninterrupted output, set the PAGESIZE to 0 and set LONG to some large number, as shown, before executing your query.)

```
set pagesize 0
set long 90000
```

```
execute DBMS_METADATA.SET_TRANSFORM_PARAM(
  DBMS_METADATA.SESSION_TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
     FROM USER_ALL_TABLES u
     WHERE u.nested='NO'
     AND (u.iot_type is null or u.iot_type='IOT');
execute DBMS_METADATA.SET_TRANSFORM_PARAM(
   DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT');
```

GET_DEPENDENT_XML and GET_DEPENDENT_DDL Functions

The GET_DEPENDENT_XML and GET_DEPENDENT_DDL functions return metadata for one or more dependent objects.

Syntax

```
DBMS_METADATA.GET_DEPENDENT_XML (
   object_type          IN VARCHAR2,
   base_object_name     IN VARCHAR2,
   base_object_schema   IN VARCHAR2 DEFAULT NULL,
   version              IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model                IN VARCHAR2 DEFAULT 'ORACLE',
   transform            IN VARCHAR2 DEFAULT NULL,
   object_count         IN NUMBER   DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_DEPENDENT_DDL (
   object_type          IN VARCHAR2,
   base_object_name     IN VARCHAR2,
   base_object_schema   IN VARCHAR2 DEFAULT NULL,
   version              IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model                IN VARCHAR2 DEFAULT 'ORACLE',
   transform            IN VARCHAR2 DEFAULT DDL,
   object_count         IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

Parameters

Table 30–16 describes the parameters for the GET_DEPENDENT_xxx function.

Table 30–16 GET_DEPENDENT_xxx Parameters

Parameter	Description
object_type	The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter. See Table 30–2, " Open() Parameters". The attributes of the object type must be appropriate to the function. For GET_DEPENDENT_xxx it must be a dependent object.
base_object_name	The base object name, which will be used internally in a BASE_OBJECT_NAME filter.
base_object_schema	The base object schema, which will be used internally in a BASE_OBJECT_SCHEMA filter. The default is the current user.
version	The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.
model	The object model to use. This parameter takes the same values as the OPEN model parameter.
transform	The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_DEPENDENT_XML this must not be DDL.
object_count	The maximum number of objects to return.

Returns

The metadata for the objects as XML or DDL.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT_NOT_FOUND. The specified object was not found in the database.

Usage Notes

The GET_DEPENDENT_xxx functions allow you to fetch metadata for dependent objects with a single call. For some object types, you can use more than one function. For example, you can use GET_xxx to fetch an index by its name or you can use GET_DEPENDENT_xxx to fetch the same index by specifying the table on which it is defined.

An arbitrary number of dependent objects may match the input criteria for GET_DEPENDENT_xxx. You can specify an object count when fetching these objects, although the default count of 10000 should usually be adequate.

If the DDL transform is specified, session-level transform parameters are inherited.

If you invoke these functions from SQL*Plus, you should use the SET LONG and SET PAGESIZE commands to generate complete, uninterrupted output.

Example: Fetch the DDL For All Object Grants On SCOTT.EMP

```
SQL> SET PAGESIZE 0
SQL> SET LONG 90000
SQL> SELECT DBMS_METADATA.GET_DEPENDENT_DDL('OBJECT_GRANT',
> 'EMP', 'SCOTT') FROM DUAL;
```

GET_GRANTED_XML and GET_GRANTED_DDL Functions

The GET_GRANTED_XML and GET_GRANTED_DDL functions return metadata for one or more granted objects.

Syntax

```
DBMS_METADATA.GET_GRANTED_XML (
   object_type          IN VARCHAR2,
   grantee              IN VARCHAR2 DEFAULT NULL,
   version              IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model                IN VARCHAR2 DEFAULT 'ORACLE',
   transform            IN VARCHAR2 DEFAULT NULL,
   object_count         IN NUMBER   DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_GRANTED_DDL (
   object_type          IN VARCHAR2,
   grantee              IN VARCHAR2 DEFAULT NULL,
   version              IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model                IN VARCHAR2 DEFAULT 'ORACLE',
   transform            IN VARCHAR2 DEFAULT DDL,
   object_count         IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

Parameters

Table 30–17 describes the parameters for the GET_GRANTED_xxx function.

Table 30–17 GET_GRANTED_xxx Parameters

Parameter	Description
object_type	The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter. See Table 30–2, " Open() Parameters". The attributes of the object type must be appropriate to the function. For GET_GRANTED_xxx it must be a granted object
grantee	The grantee. It will be used internally in a GRANTEE filter. The default is the current user.
version	The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.
model	The object model to use. This parameter takes the same values as the OPEN model parameter.
transform	The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_GRANTED_XML this must not be DDL.
object_count	The maximum number of objects to return.

Returns

The metadata for the objects as XML or DDL.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT_NOT_FOUND. The specified object was not found in the database.

Usage Notes

The GET_GRANTED_xxx functions allow you to fetch metadata for dependent objects with a single call.

An arbitrary number of granted objects may match the input criteria for GET_GRANTED_xxx. You can specify an object count when fetching these objects, although the default count of 10000 should usually be adequate.

If the DDL transform is specified, session-level transform parameters are inherited.

If you invoke these functions from SQL*Plus, you should use the SET LONG and SET PAGESIZE commands to generate complete, uninterrupted output.

Example: Fetch the DDL For All System Grants Granted to SCOTT

```
SQL> SET PAGESIZE 0
SQL> SET LONG 90000
SQL> SELECT DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT','SCOTT')
> FROM DUAL;
```

31

DBMS_MGWADM

DBMS_MGWADM defines the Messaging Gateway administrative interface. The package and object types are owned by SYS.

Note: You must run the catmgw.sql script to load the Messaging Gateway packages and types into the database. Refer to the *Oracle9i Application Developer's Guide - Advanced Queuing* for information on loading database objects.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* contains information about using DBMS_MGWADM

The following topics are discussed in this chapter:

Summary of DBMS_MGWADM Object Types and Methods

Table 31–1 DBMS_MGWADM Object Types

Object Type	Description
MGW_PROPERTY Type on page 31-2	Specifies a named property
MGW_PROPERTY.CONSTRUCT Method on page 31-3	Constructs a new MGW_PROPERTY instance
MGW_PROPERTY.CONSTRUCT Method on page 31-3	Constructs a new MGW_PROPERTY instance initialized using parameters
MGW_PROPERTIES Type on page 31-4	Specifies an array of properties
MGW_MQSERIES_PROPERTIES Type on page 31-5	Specifies basic properties for an MQSeries messaging system link
MGW_MQSERIES_PROPERTIES.CONSTRUCT Method on page 31-6	Constructs a new MGW_MQSERIES_PROPERTIES instance
MGW_MQSERIES_PROPERTIES.ALTER_CONSTRUCT Method on page 31-7	Constructs a new MGW_MQSERIES_PROPERTIES instance for altering the properties of an existing messaging link

MGW_PROPERTY Type

This type specifies a named property. MGW_PROPERTY is used to specify optional properties for messaging links and foreign queues.

Syntax

```
TYPE SYS.MGW_PROPERTY IS OBJECT(
   name VARCHAR2(100),
   value VARCHAR2(1000));
```

Attributes

Table 31–2 MGW_PROPERTY Attributes

Attribute	Description
name	Property name
value	Property value

MGW_PROPERTY.CONSTRUCT Method

This method constructs a new MGW_PROPERTY instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_PROPERTY;
```

MGW_PROPERTY.CONSTRUCT Method

This method constructs a new MGW_PROPERTY instance initialized using the given parameters.

Syntax

```
STATIC FUNCTION CONSTRUCT(
   p_name   IN VARCHAR2,
   p_value  IN VARCHAR2)
RETURN SYS.MGW_PROPERTY;
```

Parameters

Table 31–3 MGW_PROPERTY.CONSTRUCT Parameters

Parameter	Description
p_name	Property name
p_value	Property value

MGW_PROPERTIES Type

This type specifies an array of properties.

Syntax

```
TYPE SYS.MGW_PROPERTIES AS VARRAY (100) OF SYS.MGW_PROPERTY;
```

Usage Notes

Unless noted otherwise, Messaging Gateway uses named properties as follows:

- Names with the 'MGWPROP$_' prefix are reserved. They are used for special purposes and are invalid when used as a normal property name.
- A property name can exist only once in a property list; that is, a list can contain only one value for a given name. The name is treated in a case-insensitive manner.
- In general, a property list is order-independent and the property names may appear in any order. An alter property list is an exception.
- To alter an existing property list, a new property list may be used where each new property modifies the original list in one of the following ways: adds a new property, modifies a property, removes a property, or removes all properties.

The alter list is processed in order, from the first element to the last element. Thus the order in which the elements appear in the alter list is meaningful, especially when the alter list is used to remove properties from an existing list.

The property name and value are used to determine how that element affects the original list. The following rules apply:

- Add/Modify Property

```
MGW_PROPERTY.NAME = <property name>
MGW_PROPERTY.VALUE = <property value>
```

If a property of the given name already exists, the current value is replaced with the new value; otherwise the new property is added to the end of the list.

- Remove Property

```
MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE'
MGW_PROPERTY.VALUE = <name of property to remove>
```

No action is taken if the property name does not exist in the original list.

- Remove All Properties

```
MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE_ALL'
MGW_PROPERTY.VALUE = not used
```

The DBMS_MGWADM package defines constants to represent the reserved property names. Refer to the MGWPROP_< > constants.

MGW_MQSERIES_PROPERTIES Type

This type specifies basic properties for an MQSeries messaging system link.

Syntax

```
TYPE SYS.MGW_MQSERIES_PROPERTIES  IS OBJECT (
   queue_manager       VARCHAR2(64),
   hostname            VARCHAR2(64),
   port                INTEGER,
   channel             VARCHAR2(64),
   interface_type      INTEGER,
   max_connections     INTEGER,
   username            VARCHAR2(64),
   password            VARCHAR2(64),
   inbound_log_queue   VARCHAR2(64),
   outbound_log_queue  VARCHAR2(64));
```

Attributes

Table 31–4 MGW_MQSERIES_PROPERTIES Attributes

Attribute	Description
queue_manager	The name of the MQSeries queue manager
hostname	The host on which the MQSeries messaging system resides. If hostname is NULL, an MQSeries bindings connection is used. If nonnull, a client connection is used and requires that a port and channel be specified.
port	The port number. This is used only for client connections; that is, when hostname is NULL.

Table 31–4 MGW_MQSERIES_PROPERTIES Attributes

Attribute	Description
channel	The channel used when establishing a connection to the queue manager. This is used only for client connections; that is, when hostname is NULL.
interface_type	The type of messaging interface to use. Values: DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE for the MQSeries Base Java interface.
max_connections	The maximum number of messaging connections to the MQSeries messaging system
username	The user name used for authentication to the MQSeries messaging system
password	The password used for authentication to the MQSeries messaging system
inbound_log_queue	The message provider (native) name of the MQSeries queue used for propagation recovery purposes when the messaging link is used for inbound propagation; that is, when queues associated with this link serve as a propagation source. The queue must be created using MQSeries administration tools.
outbound_log_queue	The message provider (native) name of the MQSeries queue used for propagation recovery purposes when the messaging link is used for outbound propagation; that is, when queues associated with this link serve as a propagation destination. The queue must be created using MQSeries administration tools.

MGW_MQSERIES_PROPERTIES.CONSTRUCT Method

This method constructs a new MGW_MQSERIES_PROPERTIES instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_MQSERIES_PROPERTIES ;
```

MGW_MQSERIES_PROPERTIES.ALTER_CONSTRUCT Method

This method constructs a new MGW_MQSERIES_PROPERTIES instance for altering the properties of an existing messaging link. All attributes having a VARCHAR2 data type are assigned a value of DBMS_MGWADM.NO_CHANGE. Attributes of other data types are assigned a value of NULL.

Syntax

```
STATIC FUNCTION ALTER_CONSTRUCT
RETURN SYS.MGW_MQSERIES_PROPERTIES ;
```

DBMS_MGWADM Constants

Table 31–5 DBMS_MGWADM Constants—Propagation Types

Name	Type	Description
OUTBOUND_PROPAGATION	CONSTANT BINARY_INTEGER;	Represents the propagation type for AQ to non-Oracle propagation. The propagation source is a local AQ queue and the destination is a queue in a foreign (non-Oracle) messaging system.
INBOUND_PROPAGATION	CONSTANT BINARY_INTEGER;	Represents the propagation type for non-Oracle to AQ propagation. The propagation source is a queue in a foreign (non-Oracle) messaging system and the destination is a local AQ queue.

Table 31–6 DBMS_MGWADM Constants—Queue Domain Types

Name	Type	Description
DOMAIN_QUEUE	CONSTANT BINARY_INTEGER;	Represents a queue destination. A JMS queue (point-to-point model) is classified as a queue.
DOMAIN_TOPIC	CONSTANT BINARY_INTEGER;	Represents a topic destination. A JMS topic (publish-subscribe model) is classified as a topic.

Table 31–7 DBMS_MGWADM Constants—Force Values

Name	Type	Description
NO_FORCE	CONSTANT BINARY_INTEGER;	Represents a normal, nonforced action
FORCE	CONSTANT BINARY_INTEGER;	Represents a forced action

Table 31–8 DBMS_MGWADM Constants—Shutdown Modes

Name	Type	Description
SHUTDOWN_NORMAL	CONSTANT BINARY_INTEGER;	Represents the normal shutdown mode
SHUTDOWN_IMMEDIATE	CONSTANT BINARY_INTEGER;	Represents the immediate shutdown mode

Table 31–9 DBMS_MGWADM Constants—Cleanup Actions

Name	Type	Description
CLEAN_STARTUP_STATE	CONSTANT BINARY_INTEGER;	Represents the cleanup action for gateway startup state recovery

Table 31–10 DBMS_MGWADM Constants—Logging Levels

Name	Type	Description
BASIC_LOGGING	CONSTANT BINARY_INTEGER;	Represents the detail of logging information written to the log file. The logging level ranges from BASIC_LOGGING for standard (the least) information to TRACE_DEBUG_LOGGING for the greatest information.
TRACE_LITE_LOGGING	CONSTANT BINARY_INTEGER;	
TRACE_HIGH_LOGGING	CONSTANT BINARY_INTEGER;	
TRACE_DEBUG_LOGGING	CONSTANT BINARY_INTEGER;	

Table 31–11 DBMS_MGWADM Constants—MQSeries Interface Types

Name	Type	Description
MQSERIES_BASE_JAVA_INTERFACE	CONSTANT BINARY_INTEGER;	Represents the Base Java interface for the MQSeries messaging system
MQSERIES_JMS_INTERFACE	CONSTANT BINARY_INTEGER;	Represents the JMS interface for the MQSeries messaging system

Table 31–12 DBMS_MGWADM Constants—Named Property Constants

Name	Type	Description
MGWPROP_PREFIX	CONSTANT VARCHAR2;	A constant (MGWPROP$_) for the reserved property name prefix
MGWPROP_REMOVE	CONSTANT VARCHAR2;	A constant (MGWPROP$_REMOVE) for the reserved property name used to remove an existing property
MGWPROP_REMOVE_ALL	CONSTANT VARCHAR2;	A constant (MGWPROP$_REMOVE_ALL) for the reserved property name used to remove all properties

Table 31–13 DBMS_MGWADM Constants—Other Constants

Name	Type	Description
NO_CHANGE	CONSTANT VARCHAR2;	Indicates that an existing value should be preserved (not changed). This is used for certain APIs where the desire is to change one or more parameters but leave others unchanged.

MQSeries System Properties

The following sections discuss properties of MQSeries related to links and queues. Refer to IBM MQSeries documentation for more information.

Basic Link Properties (MGW_MQSERIES_PROPERTIES)

Table 31–14 summarizes the basic configuration properties for an MQSeries messaging link. (Refer to "Notes on Table 31–14" on page 31-10 for an explanation of the numbers in parentheses.) The table indicates which properties are optional (NULL allowed), which can be altered, and if alterable, which values can be dynamically changed.

Table 31–14 MQSeries Link Properties

Attribute	NULL Allowed?	Alter Value?	Dynamic?
queue_manager	no	no	--
hostname	yes (1)	no	--

Attribute	NULL Allowed?	Alter Value?	Dynamic?
port	yes (1)	no	--
channel	yes (1)	no	--
interface_type	yes (2)	no	--
max_connections	yes (3)	yes	yes
username	yes	yes	yes
password	yes	yes	yes
inbound_log_queue	yes (4)	yes(4)	yes
outbound_log_queue	yes (5)	yes(5)	yes

Notes on Table 31–14

1. If the hostname is NULL, the port and channel must be NULL. If the hostname is nonnull, the port and channel must be nonnull. If the hostname is NULL, an MQSeries bindings connection is used; otherwise a client connection is used.
2. If NULL, a default value of DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE is used.
3. If NULL, a default value of 1 is used.
4. The inbound log queue can be NULL if the link is not used for inbound propagation. The log queue can be altered only when no inbound propagation subscriber references the link.
5. The outbound log queue can be NULL if the link is not used for outbound propagation. The log queue can be altered only when no outbound propagation subscriber references the link.

Optional Link Properties

This section describes optional configuration properties supported for an MQSeries messaging link. These properties are specified by using the options parameter of DBMS_MGWADM.CREATE_MSGSYSTEM_LINK and DBMS_MGWADM.ALTER_MSGSYSTEM_LINK.

MQ_ccsid

This property specifies the character set identifier to be used. This should be the character set's integer value (for example, 819) rather than a descriptive string. If not set, the MQSeries default character set 819 is used.

Default: 819

Alterable: yes

Dynamic: no

MQ_ReceiveExit

This property specifies the fully qualified Java classname of a class implementing the MQReceiveExit interface. If not set, no default is used. This class must be in the CLASSPATH of the Messaging Gateway agent.

Default: none

Alterable: yes

Dynamic: no

MQ_SendExit

This property specifies the fully qualified Java classname of a class implementing the MQSendExit interface. If not set, no default is used. This class must be in the CLASSPATH of the Messaging Gateway agent.

Default: none

Alterable: yes

Dynamic: no

MQ_SecurityExit

This property specifies the fully qualified Java classname of a class implementing the MQSecurityExit interface. If not set, no default is used. This class must be in the CLASSPATH of the Messaging Gateway agent.

Default: none

Alterable: yes

Dynamic: no

Optional Queue Properties

This section describes optional configuration properties supported for a registered queue of an MQSeries messaging link. These properties are specified by using the `options` parameter of `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE`.

MQ_openOptions

This property specifies the value used for the `openOptions` argument of the MQSeries Base Java `MQQueueManager.accessQueue` method. No value is required but if one is given, the Messaging Gateway agent adds `MQOO_OUTPUT` to the specified value for an enqueue (`put`) operation. `MQOO_INPUT_SHARED` is added for a dequeue (`get`) operation.

Default: `MQOO_OUTPUT` for an enqueue/put operation; `MQOO_INPUT_SHARED` for a dequeue/get operation

Alterable: no

Dynamic: no

Summary of DBMS_MGWADM Subprograms

Table 31–15 DBMS_MGWADM Subprograms

Subprogram	Description
ALTER_AGENT Procedure on page 31-13	Alters Messaging Gateway agent parameters
DB_CONNECT_INFO Procedure on page 31-14	Configures connection information used by the Messaging Gateway agent for connections to the Oracle database
STARTUP Procedure on page 31-15	Starts the Messaging Gateway agent
SHUTDOWN Procedure on page 31-16	Shuts down the Messaging Gateway agent
CLEANUP_GATEWAY Procedure on page 31-17	Cleans up Messaging Gateway
SET_LOG_LEVEL Procedure on page 31-18	Dynamically alters the Messaging Gateway agent logging level
CREATE_MSGSYSTEM_LINK Procedure on page 31-18	Creates a messaging system link to an MQSeries messaging system

Table 31–15 DBMS_MGWADM Subprograms

Subprogram	Description
ALTER_MSGSYSTEM_LINK Procedure on page 31-19	Alters the properties of an MQSeries messaging system link
REMOVE_MSGSYSTEM_LINK Procedure on page 31-21	Removes a messaging system link for a non-Oracle messaging system
REGISTER_FOREIGN_QUEUE Procedure on page 31-21	Registers a non-Oracle queue entity in Messaging Gateway
UNREGISTER_FOREIGN_QUEUE Procedure on page 31-22	Removes a non-Oracle queue entity in Messaging Gateway
ADD_SUBSCRIBER Procedure on page 31-23	Adds a subscriber used to consume messages from a source queue for propagation to a destination
ALTER_SUBSCRIBER Procedure on page 31-26	Alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination
REMOVE_SUBSCRIBER Procedure on page 31-28	Removes a subscriber used to consume messages from a source queue for propagation to a destination
RESET_SUBSCRIBER Procedure on page 31-29	Resets the propagation error state for a subscriber
SCHEDULE_PROPAGATION Procedure on page 31-30	Schedules message propagation from a source to a destination
UNSCHEDULE_PROPAGATION Procedure on page 31-32	Removes a propagation schedule
ALTER_PROPAGATION_ SCHEDULE Procedure on page 31-32	Alters a propagation schedule
ENABLE_PROPAGATION_ SCHEDULE Procedure on page 31-33	Enables a propagation schedule
DISABLE_PROPAGATION_ SCHEDULE Procedure on page 31-34	Disables a propagation schedule

ALTER_AGENT Procedure

This procedure alters Messaging Gateway agent parameters.

Syntax

```
DBMS_MGWADM.ALTER_AGENT (
```

```
   max_connections  IN BINARY_INTEGER  DEFAULT NULL,
   max_memory       IN BINARY_INTEGER  DEFAULT NULL);
```

Parameters

Table 31–16 ALTER_AGENT Procedure Parameters

Parameter	Description
max_connections	The maximum number of messaging connections to the Oracle database used by the gateway agent. If NULL, the current value is unchanged. If nonnull, the value must be 1 or greater.
max_memory	The maximum heap size, in MB, used by the gateway agent. If NULL, the current value is unchanged. If nonnull, the value must be 64 or greater.

Usage Notes

The default values for configuration parameters are set when Messaging Gateway is installed.

The max_memory parameter changes take effect the next time the gateway agent is active. If the agent is currently active, the gateway must be shut down and started for the changes to take effect.

The max_connections parameter specifies the maximum number of JDBC messaging connections created and used by the AQ driver. This parameter is dynamically changed for a larger value only. In release 9.2, the gateway agent must be shut down and restarted before a smaller value takes effect.

DB_CONNECT_INFO Procedure

This procedure configures connection information used by the Messaging Gateway agent for connections to the Oracle database.

Syntax

```
DBMS_MGWADM.DB_CONNECT_INFO (
   username      IN VARCHAR2,
   password      IN VARCHAR2,
   database      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 31–17 DB_CONNECT_INFO Procedure Parameters

Parameter	Description
username	The user name used for connections to the Oracle database. NULL is not allowed
password	The password used for connections to the Oracle database. NULL is not allowed
database	The database connect string used by the gateway agent. NULL indicates that a local connection should be used.

Usage Notes

The gateway agent connects to the Oracle database as the user configured by this API. An Oracle administrator should create the user, grant it the role MGW_AGENT_ROLE, and then call this procedure to configure Messaging Gateway. The MGW_AGENT_ROLE is used to grant this user special privileges needed to access gateway configuration information stored in the database, enqueue or dequeue messages to and from Oracle queues, and perform certain AQ administration tasks.

STARTUP Procedure

This procedure starts the Messaging Gateway agent. It must be called before any propagation activity can take place.

Syntax

```
DBMS_MGWADM.STARTUP(
   instance  IN  BINARY_INTEGER DEFAULT 0,
   force     IN  BINARY_INTEGER DEFAULT dbms_mgwadm.NO_FORCE);
```

Parameters

Table 31–18 STARTUP Procedure Parameters

Parameter	Description
instance	Specifies which instance can execute the job queue job used to start the Messaging Gateway agent. If this is zero, then the job can be run by any instance.

Table 31–18 STARTUP Procedure Parameters

Parameter	Description
force	If this is dbms_mgwadm.FORCE, then any positive integer is acceptable as the job instance. If this is dbms_mgwadm.NO_FORCE (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

The Messaging Gateway agent cannot be started until an agent user has been configured using DB_CONNECT_INFO.

This procedure submits a job queue job, which starts the Messaging Gateway agent when executed. The instance and force parameters are used for job queue affinity, which you use to indicate whether a particular instance or any instance can run a submitted job.

SHUTDOWN Procedure

This procedure shuts down the Messaging Gateway agent. No propagation activity occurs until the gateway is started.

Syntax

```
DBMS_MGWADM.SHUTDOWN (
   sdmode  IN BINARY_INTEGER DEFAULT DBMS_MGWADM.SHUTDOWN_NORMAL);
```

Parameters

Table 31–19 SHUTDOWN Procedure Parameters

Parameter	Description
sdmode	The shutdown mode. Values: ■ SHUTDOWN_NORMAL for normal shutdown. The gateway agent may attempt to complete any propagation work currently in progress. ■ SHUTDOWN_IMMEDIATE for immediate shutdown. The gateway terminates any propagation work currently in progress and shuts down immediately.

Usage Notes

In release 9.2, the sdmode parameter is ignored and all shutdown modes behave the same way.

CLEANUP_GATEWAY Procedure

This procedure cleans up Messaging Gateway. The procedure performs cleanup or recovery actions that may be needed when the gateway is left in some abnormal or unexpected condition. The MGW_GATEWAY view lists gateway status and configuration information that pertains to the cleanup actions.

Syntax

```
DBMS_MGWADM.CLEANUP_GATEWAY(
   action  IN BINARY_INTEGER);
```

Parameters

Table 31–20 CLEANUP_GATWAY Procedure Parameters

Parameter	Description
action	The cleanup action to be performed. Values: CLEAN_STARTUP_STATE for gateway startup state recovery.

Usage Notes

The CLEAN_STARTUP_STATE action involves recovery tasks that set the gateway to a known state when the gateway agent has crashed or some other abnormal event occurs so that the gateway cannot be started. This should only be done when the gateway agent has been started but appears to have crashed or has been nonresponsive for an extended period of time.

Conditions or indications where this action may be needed:

- The MGW_GATEWAY view shows that the AGENT_STATUS value is something other than NOT_STARTED or START_SCHEDULED, and the AGENT_PING value is UNREACHABLE for an extended period of time.

The cleanup tasks include:

- Removing the queued job used to start the external gateway agent process.

- Setting certain configuration information to a known state. For example, setting the agent status to NOT_STARTED.

The following considerations apply:

- This fails if the agent status is NOT_STARTED or START_SCHEDULED.
- This fails if no shutdown attempt has been made prior to calling this procedure, except if the agent status is STARTING.
- This attempts to contact (ping) the gateway agent. If successful, the assumption is that the agent is active and this procedure fails. If the agent does not respond after several attempts have been made, the cleanup tasks are performed.
- This procedure takes several seconds, possibly up to one minute, if the gateway agent never responds to the ping attempts. This is expected behavior under conditions where this particular cleanup action is appropriate and necessary.

SET_LOG_LEVEL Procedure

This procedure dynamically alters the Messaging Gateway agent logging level. The Messaging Gateway agent must be running.

Syntax

```
DBMS_MGWADM.SET_LOG_LEVEL (
   log_level  IN BINARY_INTEGER);
```

Parameters

Table 31–21 SET_LOG_LEVEL Procedure Parameters

Parameter	Description
log_level	Level at which the Messaging Gateway agent logs information; refer to the DBMS_MGWADM.<>_LOGGING constants. BASIC_LOGGING generates the least information while TRACE_DEBUG_LOGGING generates the most information.

CREATE_MSGSYSTEM_LINK Procedure

This procedure creates a messaging system link to an MQSeries messaging system.

Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
   linkname    IN VARCHAR2,
   properties  IN sys.mgw_mqseries_properties,
   options     IN sys.mgw_properties DEFAULT NULL,
   comment     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 31–22 CREATE_MSGSYSTEM_LINK Procedure Parameters

Parameter	Description
linkname	A user-defined name to identify the message system link
properties	Basic properties of an MQSeries messaging system link
options	Optional link properties. NULL if there are none. These are less frequently used configuration properties supported by the messaging system.
comment	A user-specified description. NULL if one is not desired

Usage Notes

Refer to "Basic Link Properties (MGW_MQSERIES_PROPERTIES)" on page 31-9 for more information about messaging link properties.

ALTER_MSGSYSTEM_LINK Procedure

This procedure alters the properties of an MQSeries messaging system link.

Syntax

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (
   linkname   IN  VARCHAR2,
   properties IN  SYS.MGW_MQSERIES_PROPERTIES,
   options    IN  SYS.MGW_PROPERTIES DEFAULT NULL,
   comment    IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE);
```

Parameters

Table 31–23 ALTER_MSGSYSTEM_LINK Procedure Parameters

Parameters	Description
linkname	The messaging system link name
properties	Basic properties for an MQSeries messaging system link. If NULL, no link properties are changed.
options	Optional link properties. NULL if no options are changed. If nonnull, the properties specified in this list are combined with the current options properties to form a new set of link options.
comment	An optional description or NULL if not desired. If DBMS_MGWADM.NO_CHANGE is specified, the current value is not changed.

Usage Notes

In release 9.2, the MGW_MQSERIES_PROPERTIES.MAX_CONNECTIONS parameter specifies the maximum number of messaging connections created and used for that messaging link. This parameter is dynamically changed for a larger value only. The gateway agent must be shut down and restarted before a smaller value takes effect.

To retain an existing value for a messaging link property with a VARCHAR2 data type, specify DBMS_MGWADM.NO_CHANGE for that particular property. To preserve an existing value for a property of another data type, specify NULL for that property.

The options parameter specifies a set of properties used to alter the current option properties. Each property affects the current property list in a particular manner; add a new property, replace an existing property, remove an existing property, or remove all properties.

Some properties cannot be modified and this procedure will fail if you try. Other properties can be modified only under certain conditions, depending on the current configuration; for example, when there are no propagation subscribers or schedules that have a source or destination associated with the link.

For properties that can be changed, a few are dynamic, while others require Messaging Gateway to be shut down and restarted before they take effect.

Refer to "Basic Link Properties (MGW_MQSERIES_PROPERTIES)" on page 31-9 for more information on messaging link properties.

REMOVE_MSGSYSTEM_LINK Procedure

This procedure removes a messaging system link for a non-Oracle messaging system.

Syntax

```
DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK(
   linkname  IN VARCHAR2);
```

Parameters

Table 31–24 REMOVE_MSGSYSTEM_LINK Procedure Parameters

Parameters	Description
linkname	The messaging system link name

Usage Notes

All registered queues associated with this link must be removed before the messaging system link can be removed. This fails if there is a registered foreign (non-Oracle) queue that references this link.

REGISTER_FOREIGN_QUEUE Procedure

This procedure registers a non-Oracle queue entity in Messaging Gateway.

Syntax

```
DBMS_MGWADM.REGISTER_FOREIGN_QUEUE(
   name            IN VARCHAR2,
   linkname        IN VARCHAR2,
   provider_queue  IN VARCHAR2 DEFAULT NULL,
   domain          IN INTEGER DEFAULT NULL,
   options         IN sys.mgw_properties DEFAULT NULL,
   comment         IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 31–25 REGISTER_FOREIGN_QUEUE Procedure Parameters

Parameters	Description
name	The registered queue name. This name identifies the foreign queue within Messaging Gateway and need not match the name of the queue in the foreign messaging system.
linkname	The link name for the messaging system on which this queue exists
provider_queue	The message provider (native) queue name. If NULL, the value provided for the name parameter is used as the provider queue name.
domain	The domain type of the queue. Values: ■ NULL if the domain type is automatically determined based on the messaging system of the queue ■ DOMAIN_QUEUE for a queue (point-to-point model) ■ DOMAIN_TOPIC for a topic (publish-subscribe model)
options	Optional queue properties
comment	A user-specified description. Can be NULL.

Usage Notes

This procedure does not create the physical queue in the non-Oracle messaging system. The non-Oracle queue must be created using the administration tools for that messaging system.

In release 9.2, domain is not used and must be NULL because the domain type can be automatically determined for the messaging systems currently supported.

Refer to "Basic Link Properties (MGW_MQSERIES_PROPERTIES)" on page 31-9 for more information on messaging link properties.

UNREGISTER_FOREIGN_QUEUE Procedure

This procedure removes a non-Oracle queue entity in Messaging Gateway.

Syntax

```
DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE(
```

```
   name         IN VARCHAR2,
   linkname     IN VARCHAR2);
```

Parameters

Table 31–26 UNREGISTER_FOREIGN_QUEUE Procedure Parameters

Parameter	Description
name	The queue name
linkname	The link name for the messaging system on which the queue exists

Usage Notes

This procedure does not remove the physical queue in the non-Oracle messaging system.

All subscribers and schedules referencing this queue must be removed before it can be unregistered. This fails if a subscriber or propagation schedule references the non-Oracle queue.

ADD_SUBSCRIBER Procedure

This procedure adds a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.ADD_SUBSCRIBER(
   subscriber_id     IN VARCHAR2,
   propagation_type  IN BINARY_INTEGER,
   queue_name        IN VARCHAR2,
   destination       IN VARCHAR2,
   rule              IN VARCHAR2 DEFAULT NULL,
   transformation    IN VARCHAR2 DEFAULT NULL,
   exception_queue   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 31–27 ADD_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Specifies a user-defined name that identifies this subscriber.
propagation_type	Specifies the type of message propagation. Values: ■ DBMS_MGWADM.OUTBOUND_PROPAGATION for AQ to non-Oracle propagation ■ DBMS_MGWADM.INBOUND_PROPAGATION for non-Oracle to AQ propagation
queue_name	Specifies the source queue to which this subscriber is being added. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
destination	Specifies the destination queue to which messages consumed by this subscriber are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. This is NULL if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
transformation	Specifies the transformation needed to convert between the AQ payload and a gateway-defined ADT. The type of transformation needed depends on the value specified for propagation_type. If no transformation is provided (a NULL value is specified), the gateway makes a best effort to propagate messages based on the AQ payload type and the capabilities of the non-Oracle messaging system. For example, the gateway automatically propagates messages for an AQ queue having a RAW payload and non-Oracle messaging systems that support a 'bytes' message body.
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, an exception queue is not used and propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type. The source queue and exception queue cannot be the same queue.

Usage Notes

For OUTBOUND_PROPAGATION, parameters are interpreted as follows:

- queue_name - specifies the local AQ queue that is the propagation source. This must have a syntax of schema.queue.
- destination - specifies the foreign queue to which messages are propagated. This must have a syntax of registered_queue@message_link.
- rule - specifies an optional AQ subscriber rule. This is NULL if no rule is needed.
- transformation - specifies the transformation used to convert the AQ payload to a gateway-defined ADT.

  The gateway propagation dequeues messages from the AQ queue using the transformation to convert the AQ payload to a known gateway-defined ADT. The message is then enqueued in the foreign messaging system based on the gateway ADT.
- exception_queue - specifies the name of a local AQ queue to which messages are moved if an exception occurs. This must have a syntax of schema.queue.

For INBOUND_PROPAGATION, parameters are interpreted as follows:

- queue_name - specifies the foreign queue that is the propagation source. This must have a syntax of registered_queue@message_link.
- destination - specifies the local AQ queue to which message are propagated. This must have a syntax of schema.queue.
- rule - specifies an optional subscriber rule that is valid for the foreign messaging system. This is NULL if no rule is needed.
- transformation - specifies the transformation used to convert a gateway-defined ADT to the AQ payload type.

  The gateway propagation dequeues messages from the foreign messaging system and converts the message body to a known gateway-defined ADT. The transformation is used to convert the gateway ADT to an AQ payload type when the message is enqueued to the AQ queue.
- exception_queue - specifies the name of a foreign queue to which messages are moved if an exception occurs. This must have a syntax of registered_queue@message_link.

For OUTBOUND_PROPAGATION, a local subscriber is added to the AQ queue. The subscriber is of the form aq$_agent('MGW_<subscriber_id>',NULL,NULL).

For INBOUND_PROPAGATION, whether or not a subscriber is needed depends on the requirements of the non-Oracle messaging system.

For OUTBOUND_PROPAGATION, the exception queue has the following considerations:

- The user is responsible for creating the AQ queue to be used as the exception queue.
- The payload type of the source and exception queue must match.
- The exception queue must be created as a queue type of NORMAL_QUEUE rather than EXCEPTION_QUEUE. Enqueue restrictions prevent the gateway propagation from using an AQ queue of type EXCEPTION_QUEUE as a gateway exception queue.

For INBOUND_PROPAGATION, the exception queue has the following considerations:

- The exception queue must be a registered non-Oracle queue.
- The source and exception queues must use the same messaging system link.

ALTER_SUBSCRIBER Procedure

This procedure alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.ALTER_SUBSCRIBER (
   subscriber_id    IN VARCHAR2,
   rule             IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
   transformation   IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
   exception_queue  IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

Parameters

Table 31–28 ALTER_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Identifies the subscriber to be altered
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. The syntax and interpretation of this parameter depend on the subscriber's propagation type. A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.
transformation	Specifies the transformation needed to convert between the AQ payload and a gateway-defined ADT. The type of transformation needed depends on the subscriber's propagation type. A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If no exception queue is associated with the subscriber, propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the subscriber's propagation type. A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged. The source queue and exception queue cannot be the same queue.

Usage Notes

For a subscriber having a propagation type of OUTBOUND_PROPAGATION, parameters are interpreted as follows:

- rule - specifies an optional AQ subscriber rule.
- transformation - specifies the transformation used to convert the AQ payload to a gateway-defined ADT.

  The gateway propagation dequeues messages from the AQ queue using the transformation to convert the AQ payload to a known gateway-defined ADT. The message is then enqueued in the foreign messaging system based on the gateway ADT.

- exception_queue - specifies the name of a local AQ queue to which messages are moved if an exception occurs. This must have a syntax of schema.queue.

For a subscriber having a propagation type of INBOUND_PROPAGATION, parameters are interpreted as follows:

- rule - specifies an optional subscriber rule that is valid for the foreign messaging system.
- transformation - specifies the transformation used to convert a gateway-defined ADT to the AQ payload type.

  The gateway propagation dequeues messages from the foreign messaging system and converts the message body to a known gateway-defined ADT. The transformation is used to convert the gateway ADT to an AQ payload type when the message is enqueued to the AQ queue.
- exception_queue - specifies the name of a foreign queue to which messages are moved if an exception occurs. This must have a syntax of registered_queue@message_link.

For OUTBOUND_PROPAGATION, the exception queue has the following considerations:

- The user is responsible for creating the AQ queue to be used as the exception queue.
- The payload type of the source and exception queues must match.
- The exception queue must be created as a queue type of NORMAL_QUEUE rather than EXCEPTION_QUEUE. Enqueue restrictions prevent gateway propagation from using an AQ queue of type EXCEPTION_QUEUE as a gateway exception queue.

For INBOUND_PROPAGATION, the exception queue has the following considerations:

- The exception queue must be a registered non-Oracle queue.
- The source and exception queues must use the same messaging system link.

REMOVE_SUBSCRIBER Procedure

This procedure removes a subscriber used to consume messages from a source queue for propagation to a destination.

Syntax

```
DBMS_MGWADM.REMOVE_SUBSCRIBER (
   subscriber_id  IN VARCHAR2,
   force          IN BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE );
```

Parameters

Table 31–29 REMOVE_SUBSCRIBER Procedure Parameters

Parameter	Description
`subscriber_id`	Identifies the subscriber to be removed
`force`	Specifies whether this procedure should succeed even if the gateway is not able to perform all cleanup actions pertaining to this subscriber. Values: ▪ `NO_FORCE (0)` if this should fail if unable to cleanup successfully. ▪ `FORCE (1)` if this should succeed even though all cleanup actions may not be done. The gateway agent uses various resources of the Oracle database and non-Oracle messaging system for its propagation work; for example, it enqueues messages to log queues, creates subscribers, and so on. These resources are typically associated with each subscriber and need to be released when the subscriber is no longer needed. Typically, this procedure should only be called when the gateway agent is running and able to access the non-Oracle messaging system associated with this subscriber.

Usage Notes

For outbound propagation, a local subscriber is removed from the AQ queue.

RESET_SUBSCRIBER Procedure

This procedure resets the propagation error state for a subscriber.

Syntax

```
DBMS_MGWADM.RESET_SUBSCRIBER (
   subscriber_id  IN VARCHAR2 );
```

Parameters

Table 31–30 RESET_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_id	Identifies the subscriber

SCHEDULE_PROPAGATION Procedure

This procedure schedules message propagation from a source to a destination. The schedule must be enabled and the gateway started in order for messages to be propagated.

Syntax

```
DBMS_MGWADM.SCHEDULE_PROPAGATION (
   schedule_id       IN VARCHAR2,
   propagation_type  IN BINARY_INTEGER,
   source            IN VARCHAR2,
   destination       IN VARCHAR2,
   start_time        IN DATE DEFAULT SYSDATE,
   duration          IN NUMBER DEFAULT NULL,
   next_time         IN VARCHAR2 DEFAULT NULL,
   latency           IN NUMBER DEFAULT 60 );
```

Parameters

Table 31–31 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
schedule_id	Specifies a user-defined name that identifies the schedule.
propagation_type	Specifies the type of message propagation. Values: ▪ DBMS_MGWADM.OUTBOUND_PROPAGATION for AQ to non-Oracle propagation ▪ DBMS_MGWADM.INBOUND_PROPAGATION for non-Oracle to AQ propagation.
source	Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.

Table 31–31 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
destination	Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
start_time	Specifies the initial start time for the propagation window for messages from the source queue to the destination
duration	Specifies the duration of the propagation window, in seconds. A NULL value means that the propagation window is forever, or until the propagation is unscheduled.
next_time	Specifies the date function to compute the start of the next propagation window from the end of the current window. A NULL value means that the propagation is stopped at the end of the current window.
latency	Specifies the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. However, if for example, the latency is 60 seconds, and if no messages are waiting to be propagated, then during the propagation window, no messages are propagated from the source to the destination for at least 60 more seconds. If the latency is 0, then a message is propagated as soon as it is enqueued.

Usage Notes

In release 9.2, all window parameters are ignored.

For OUTBOUND_PROPAGATION, parameters are as follows:

- source - specifies the local AQ queue, which is the propagation source. This must have a syntax of schema.queue.
- destination - specifies the foreign queue to which messages are propagated. This must have a syntax of registered_queue@message_link.

For INBOUND_PROPAGATION, parameters are interpreted as follows:

- source - specifies the foreign queue, which is the propagation source. This must have a syntax of registered_queue@message_link.
- destination - specifies the local AQ queue to which message are propagated. This must have a syntax of schema.queue.

The schedule is set to an enabled state when it is created.

UNSCHEDULE_PROPAGATION Procedure

This procedure removes a propagation schedule.

Syntax

```
DBMS_MGWADM.UNSCHEDULE_PROPAGATION (
   schedule_id   IN VARCHAR2 );
```

Parameters

Table 31–32 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be removed

ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters a propagation schedule.

Syntax

```
DBMS_MGWADM.ALTER_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2,
   duration     IN NUMBER DEFAULT NULL,
   next_time    IN VARCHAR2 DEFAULT NULL,
   latency      IN NUMBER DEFAULT 60 );
```

Parameters

Table 31–33 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be altered
duration	Specifies the duration of the propagation window, in seconds. A NULL value means that the propagation window is forever, or until the propagation is unscheduled.

Table 31–33 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
next_time	Specifies the date function to compute the start of the next propagation window from the end of the current window. A NULL value means that the propagation is stopped at the end of the current window.
latency	Specifies the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. However, if for example, the latency is 60 seconds, and if no messages are waiting to be propagated, then during the propagation window, no messages are propagated from the source to the destination for at least 60 additional seconds. If the latency is 0, then a message is propagated as soon as it is enqueued.

Usage Notes

In release 9.2, propagation window parameters are ignored.

Caution: This procedure always overwrites the existing value for each parameter. If a given parameter is not specified, the existing values are overwritten with the default value.

ENABLE_PROPAGATION_SCHEDULE Procedure

This procedure enables a propagation schedule.

Syntax

```
DBMS_MGWADM.ENABLE_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2 );
```

Parameters

Table 31–34 ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be enabled

DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE (
   schedule_id  IN VARCHAR2 );
```

Parameters

Table 31–35 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
schedule_id	Identifies the propagation schedule to be disabled

Summary of Database Views

The views listed in Table 31–36 provide Messaging Gateway configuration, status, and statistical information. Unless otherwise indicated, the SELECT privilege is granted to MGW_ADMINISTRATOR_ROLE so that only Messaging Gateway administrators have access to the views. All views are owned by SYS.

Table 31–36 Database Views

Name	Description
MGW_GATEWAY View	Lists configuration and status information for Messaging Gateway
MGW_LINKS View	Lists the name and types of messaging system links currently created
MGW_MQSERIES_LINKS View	Lists messaging system properties for MQSeries links

Table 31–36 Database Views

Name	Description
MGW_FOREIGN_QUEUES View	Lists the queue properties of registered queues
MGW_SUBSCRIBERS View	Lists subscriber properties, status, and statistical information
MGW_SCHEDULES View	Lists schedule properties and status

MGW_GATEWAY View

This view lists configuration and status information for Messaging Gateway, as shown in Table 31–37.

Table 31–37 MGW_GATEWAY Properties

Name	Type	Description
AGENT_STATUS	VARCHAR2	Status of the gateway agent. Values: ■ NOT_STARTED if the gateway agent has not been started. ■ START_SCHEDULED if gateway agent has been scheduled to start; this indicates the gateway has been started using DBMS_MGWADM.STARTUP but the queued job used to start the gateway agent has not been executed. ■ STARTING if gateway agent is starting; this indicates the queued job has been executed and the gateway agent is starting up. ■ INITIALIZING if gateway agent has started and is initializing. ■ RUNNING if gateway agent is running. ■ SHUTTING_DOWN if gateway agent is shutting down.
AGENT_PING	VARCHAR2	Gateway agent ping status. Values: ■ NULL if no ping attempt was made. ■ REACHABLE if ping attempt was successful. ■ UNREACHABLE if ping attempt failed. AGENT_PING attempts to contact the gateway agent. There is a short delay (up to 5 seconds) if the ping attempt fails. No ping is attempted if the AGENT_STATUS is NOT_STARTED or START_SCHEDULED.
AGENT_JOB	NUMBER	Job number of the queued job used to start the Messaging Gateway agent process. The job number is set when the Messaging Gateway is started and cleared when it shuts down.

Table 31–37 MGW_GATEWAY Properties

Name	Type	Description
AGENT_USER	VARCHAR2	Database user name used by the gateway agent to connect to the database
AGENT_DATABASE	VARCHAR2	The database connect string used by the gateway agent. NULL indicates that a local connection is used.
LAST_ERROR_DATE	DATE	Date of last Messaging Gateway agent error. The last error information is cleared when Messaging Gateway is started. It is set if the Messaging Gateway agent fails to start or terminates due to an abnormal condition.
LAST_ERROR_TIME	VARCHAR2	Time of last Messaging Gateway agent error
LAST_ERROR_MSG	VARCHAR2	Message for last Messaging Gateway agent error
MAX_CONNECTIONS	NUMBER	Maximum number of messaging connections to the Oracle database
MAX_MEMORY	NUMBER	Maximum heap size used by gateway agent (in MB)

MGW_LINKS View

This view lists the names and types of messaging system links currently defined.

Table 31–38 MGW_LINKS Properties

Name	Type	Description
LINK_NAME	VARCHAR2	Name of the messaging system link
LINK_TYPE	VARCHAR2	Type of messaging system link. Values: MQSERIES
LINK_COMMENT	VARCHAR2	User comment for the link

MGW_MQSERIES_LINKS View

This view lists information for the MQSeries messaging system links. The view includes most of the messaging system properties specified when the link is created.

Table 31–39 MGW_MQSERIES_LINKS Properties

Name	Type	Description
LINK_NAME	VARCHAR2	Name of the messaging system link
QUEUE_MANAGER	VARCHAR2	Name of the MQSeries queue manager
HOSTNAME	VARCHAR2	Name of the MQSeries host
PORT	NUMBER	Port number
CHANNEL	VARCHAR2	Connection channel
INTERFACE_TYPE	VARCHAR2	Messaging interface type. Values: BASE_JAVA for the MQSeries Base Java interface
MAX_CONNECTIONS	NUMBER	Maximum number of messaging connections
INBOUND_LOG_QUEUE	VARCHAR2	Inbound propagation log queue
OUTBOUND_LOG_QUEUE	VARCHAR2	Outbound propagation log queue
OPTIONS	SYS.MGW.PROPERTIES	Link options
LINK_COMMENT	VARCHAR2	User comment for the link

MGW_FOREIGN_QUEUES View

This view lists information for foreign queues. The view includes most of the queue properties specified when the queue is registered.

Table 31–40 MGW_FOREIGN_QUEUES Properties

Name	Type	Description
NAME	VARCHAR2	Name of the registered queue
LINK_NAME	VARCHAR2	Name of the messaging system link
PROVIDER_QUEUE	VARCHAR2	Message provider (native) queue name

Table 31–40 MGW_FOREIGN_QUEUES Properties

Name	Type	Description
DOMAIN	VARCHAR2	Queue domain type. Values: ■ NULL if automatically determined by messaging system ■ QUEUE for a queue (point-to-point) model ■ TOPIC for a topic (publish-subscribe) model
OPTIONS	SYS.MGW.PROPERTIES	Optional queue properties
QUEUE_COMMENT	VARCHAR2	User comment for the foreign queue

MGW_SUBSCRIBERS View

This view lists configuration and status information for Messaging Gateway subscribers. The view includes most of the subscriber properties specified when the subscriber is added, as well as other status and statistical information.

Table 31–41 MGW_SUBSCRIBERS Properties

Name	Type	Description
SUBSCRIBER_ID	VARCHAR2	Propagation subscriber identifier
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: ■ OUTBOUND for AQ to non-Oracle propagation ■ INBOUND for non-Oracle to AQ propagation
QUEUE_NAME	VARCHAR2	Subscriber source queue
DESTINATION	VARCHAR2	Destination queue to which messages are propagated
RULE	VARCHAR2	Subscription rule
TRANSFORMATION	VARCHAR2	Transformation used for message conversion
EXCEPTION_QUEUE	VARCHAR2	Exception queue used for logging purposes

Table 31–41 MGW_SUBSCRIBERS Properties

Name	Type	Description
STATUS	VARCHAR2	Subscriber status. Values: ■ ENABLED if the subscriber is enabled ■ DELETE_PENDING if subscriber removal is pending; typically the case when DBMS_MGWADM.REMOVE_SUBSCRIBER has been called but certain cleanup tasks pertaining to this subscriber are still outstanding.
FAILURES	NUMBER	Number of propagation failures
LAST_ERROR_DATE	DATE	Date of last propagation error
LAST_ERROR_TIME	VARCHAR2	Time of last propagation error
LAST_ERROR_MSG	VARCHAR2	Message for last propagation error
PROPAGATED_MSGS	NUMBER	Number of messages propagated to the destination queue since the last time the agent was started
EXCEPTIONQ_MSGS	NUMBER	Number of messages moved to the propagation exception queue since the last time the agent was started

MGW_SCHEDULES View

This view lists configuration and status information for Messaging Gateway schedules. The view includes most of the schedule properties specified when the schedule is created, as well as other status information.

Table 31–42 MGW_SCHEDULES Properties

Name	Type	Description
SCHEDULE_ID	VARCHAR2	Propagation schedule identifier
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: ■ OUTBOUND for AQ to non-Oracle propagation ■ INBOUND for non-Oracle to AQ propagation
SOURCE	VARCHAR2	Propagation source
DESTINATION	VARCHAR2	Propagation destination
START_DATE	DATE	Schedule start date

Table 31–42 MGW_SCHEDULES Properties

Name	Type	Description
START_TIME	VARCHAR2	Schedule start time
PROPAGATION_WINDOW	NUMBER	Duration of the propagation window (in seconds)
NEXT_TIME	VARCHAR2	Date function used to compute the start of the next propagation window
LATENCY	NUMBER	Propagation window latency (in seconds)
SCHEDULE_DISABLED	VARCHAR2	Indicates whether the schedule is disabled. Values: ■ Y if schedule is disabled ■ N if schedule is enabled

32

DBMS_MGWMSG

DBMS_MGWMSG provides object types—used by the canonical message types to convert message bodies—and helper methods, constants, and subprograms for working with the Messaging Gateway message types. The package and object types are owned by SYS.

Note: You must run the catmgw.sql script to load the Messaging Gateway packages and types into the database. Refer to the *Oracle9i Application Developer's Guide - Advanced Queuing* for information on loading database objects.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* contains information about using DBMS_MGWMSG.

The following topics are discussed in this chapter:

Summary of DBMS_MGWMSG Object Types and Methods

Table 32–1 DBMS_MGWMSG Object Types and Methods

Object Type	Description
MGW_NAME_VALUE_T Type	Specifies a named value
MGW_NAME_VALUE_T.CONSTRUCT Method	Constructs a new MGW_NAME_VALUE_T instance
MGW_NAME_VALUE_T.CONSTRUCT_ <TYPE> Methods	Constructs a new MGW_NAME_VALUE_T instance initialized with the value of a specific type
MGW_NAME_TYPE_ARRAY_T Type	Specifies an array of name-value pairs
MGW_TEXT_VALUE_T Type	Specifies a TEXT value
MGW_TEXT_VALUE_T.CONSTRUCT Method	Constructs a new MGW_TEXT_VALUE_T instance
MGW_RAW_VALUE_T Type	Specifies a RAW value
MGW_RAW_VALUE_T.CONSTRUCT Method	Constructs a new MGW_RAW_VALUE_T instance
MGW_BASIC_MSG_T Type	A canonical type for a basic TEXT or RAW message
MGW_BASIC_MSG_T.CONSTRUCT Method	Constructs a new MGW_BASIC_MSG_T instance

MGW_NAME_VALUE_T Type

This type specifies a named value. The name and type attributes and one of the < >_value attributes are typically nonnull.

Syntax

```
TYPE SYS.MGW_NAME_VALUE_T IS OBJECT
   name              VARCHAR2(250),
   type              INTEGER,
   integer_value     INTEGER,
   number_value      NUMBER,
   text_value        VARCHAR2(4000),
   raw_value         RAW(2000),
   date_value        DATE);
```

Attributes

Table 32–2 MGW_NAME_VALUE_T Attributes

Attribute	Description
name	Name associated with the value
type	Value type. Refer to the DBMS_MGWMSG.< >_VALUE constants in Table 32–7. This indicates which Java datatype and class are associated with the value. It also indicates which attribute stores the value.
integer_value	Stores a numeric integer value
number_value	Stores a numeric float or large integer value
text_value	Stores a TEXT value
raw_value	Stores a RAW (bytes) value
date_value	Stores a date value

Type-Attribute Mapping

Table 32–3 shows the mapping between the value type and the attribute used to store the value.

Table 32–3 Type-Attribute Mapping

Type	Value Stored in Attribute
DBMS_MGWMSG.TEXT_VALUE	text_value
DBMS_MGWMSG.RAW_VALUE	raw_value
DBMS_MGWMSG.BOOLEAN_VALUE	integer_value
DBMS_MGWMSG.BYTE_VALUE	integer_value
DBMS_MGWMSG.SHORT_VALUE	integer_value
DBMS_MGWMSG.INTEGER_VALUE	integer_value
DBMS_MGWMSG.LONG_VALUE	number_value
DBMS_MGWMSG.FLOAT_VALUE	number_value
DBMS_MGWMSG.DOUBLE_VALUE	number_value
DBMS_MGWMSG.DATE_VALUE	date_value

MGW_NAME_VALUE_T.CONSTRUCT Method

This method constructs a new MGW_NAME_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_NAME_VALUE_T;
```

MGW_NAME_VALUE_T.CONSTRUCT_<TYPE> Methods

These methods construct a new MGW_NAME_VALUE_T instance initialized with the value of a specific type. Each method sets the name and type attributes and one of the < >_value attributes, as shown in the mappings in Table 32–3.

Syntax

```
STATIC FUNCTION CONSTRUCT_BOOLEAN (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_BYTE (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_SHORT (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_INTEGER (
   name   IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_LONG (
   name   IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,
```

```
STATIC FUNCTION CONSTRUCT_FLOAT (
   name   IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DOUBLE (
   name   IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_TEXT (
   name   IN VARCHAR2,
   value  IN VARCHAR2 )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_RAW (
   name  IN VARCHAR2,
   value  IN RAW )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DATE (
   name   IN VARCHAR2,
   value  IN DATE )
RETURN SYS.MGW_NAME_VALUE_T );
```

Usage Notes

The `construct_boolean` method sets the value to either `DBMS_MGWMSG.BOOLEAN_TRUE` or `DBMS_MGWMSG.BOOLEAN_FALSE`.

MGW_NAME_TYPE_ARRAY_T Type

This type specifies an array of name-value pairs. An object of `MGW_NAME_VALUE_ARRAY_T` type can have up to 1024 elements.

Syntax

```
TYPE SYS.MGW_NAME_VALUE_ARRAY_T AS VARRAY (1024) OF SYS.MGW_NAME_VALUE_T;
```

MGW_TEXT_VALUE_T Type

This type specifies a TEXT value. It can store a large value as a CLOB or a smaller value (size <= 4000) as VARCHAR2. Only one of the < >_ value attributes should be set.

Syntax

```
TYPE SYS.MGW_TEXT_VALUE_T IS OBJECT
   small_value VARCHAR2(4000),
   large_value CLOB);
```

Attributes

Table 32–4 MGW_TEXT_VALUE_T Attributes

Attribute	Description
small_value	Small TEXT value. Used for values <= 4000.
large_value	Large TEXT value. Used when the value is too large for the small_value attribute.

MGW_TEXT_VALUE_T.CONSTRUCT Method

This method constructs a new MGW_TEXT_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T;
```

MGW_RAW_VALUE_T Type

This type specifies a RAW value. This type can store a large value as a BLOB or a smaller value (size <= 2000) as RAW. Only one of the < >_value attributes should be set.

Syntax

```
TYPE SYS.MGW_RAW_VALUE_T IS OBJECT(
   small_value RAW(2000),
   large_value BLOB);
```

Attributes

Table 32–5 MGW_RAW_VALUE_T Attributes

Attribute	Description
small_value	Small RAW (bytes) value <= 2000
large_value	Large RAW value. Used when the value is too large for the small_value attribute.

MGW_RAW_VALUE_T.CONSTRUCT Method

This method constructs a new MGW_RAW_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_RAW_VALUE_T;
```

MGW_BASIC_MSG_T Type

This is a canonical type for a basic TEXT or RAW message. Only a single TEXT or RAW value is typically set. An object of this type should not have both TEXT and RAW set to a nonnull value at the same time.

Syntax

```
TYPE SYS.MGW_BASIC_MSG_T IS OBJECT
   header     SYS.MGW_NAME_VALUE_ARRAY_T,
   text_body  SYS.MGW_TEXT_VALUE_T,
   raw_body   SYS.MGW_RAW_VALUE_T);
```

Attributes

Table 32–6 MGW_BASIC_MSG_T Attributes

Attribute	Description
header	Message header information as an array of name-value pairs
text_body	Message body for a TEXT message
raw_body	Message body for a RAW (bytes) message

MGW_BASIC_MSG_T.CONSTRUCT Method

This method constructs a new MGW_BASIC_MSG_T instance. All attributes are assigned a value of NULL.

Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T;
```

DBMS_MGWMSG Constants

Table 32–7 DBMS_MGWMSG Constants: Value Types—Constants representing the type of value for a SYS.MGW_NAME_VALUE_T object

Value	Constant
TEXT_VALUE	CONSTANT BINARY_INTEGER := 1;
RAW_VALUE	CONSTANT BINARY_INTEGER := 2;
BOOLEAN_VALUE	CONSTANT BINARY_INTEGER := 3;
BYTE_VALUE	CONSTANT BINARY_INTEGER := 4;
SHORT_VALUE	CONSTANT BINARY_INTEGER := 5;
INTEGER_VALUE	CONSTANT BINARY_INTEGER := 6;
LONG_VALUE	CONSTANT BINARY_INTEGER := 7;
FLOAT_VALUE	CONSTANT BINARY_INTEGER := 8;
DOUBLE_VALUE	CONSTANT BINARY_INTEGER := 9;
DATE_VALUE	CONSTANT BINARY_INTEGER := 10;

Table 32–8 DBMS_MGWMSG Constants: Boolean Values—Constants Representing a Boolean as a Numeric Value

Value	Constant
BOOLEAN_FALSE	CONSTANT BINARY_INTEGER := 0;
BOOLEAN_TRUE	CONSTANT BINARY_INTEGER := 1;

Table 32–9 DBMS_MGWMSG Constants: Case Comparisons

Value	Constant
CASE_SENSITIVE	CONSTANT BINARY_INTEGER := 0;
CASE_INSENSITIVE	CONSTANT BINARY_INTEGER := 1;

Summary of DBMS_MGWMSG Subprograms

Table 32–10 DBMS_MGWMSG Subprograms

Subprogram	Description
NVARRAY_ADD Procedure on page 32-10	Appends a name-value element to the end of a name-value array
NVARRAY_GET Function on page 32-10	Gets the name-value element of the name you specify in p_name from a name-value array
NVARRAY_GET_BOOLEAN Function on page 32-11	Gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type
NVARRAY_GET_BYTE Function on page 32-12	Gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type
NVARRAY_GET_SHORT Function on page 32-13	Gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type
NVARRAY_GET_INTEGER Function on page 32-13	Gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type
NVARRAY_GET_LONG Function on page 32-14	Gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type
NVARRAY_GET_FLOAT Function on page 32-15	Gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type
NVARRAY_GET_DOUBLE Function on page 32-15	Gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type

Table 32–10 DBMS_MGWMSG Subprograms

Subprogram	Description
NVARRAY_GET_TEXT Function on page 32-16	Gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type
NVARRAY_GET_RAW Function on page 32-17	Gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type
NVARRAY_GET_DATE Function on page 32-17	Gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type
NVARRAY_FIND_NAME Function on page 32-18	Searches a name-value array for the element with the name you specify in p_name
NVARRAY_FIND_NAME_TYPE Function on page 32-19	Searches a name-value array for an element with the name and value type you specify

NVARRAY_ADD Procedure

This procedure appends a name-value element to the end of a name-value array.

Syntax

```
DBMS_MGWMSG.NVARRAY_ADD (
   p_array  IN OUT SYS.MGW_NAME_VALUE_ARRAY_T,
   p_value  IN     SYS.MGW_NAME_VALUE_T );
```

Parameters

Table 32–11 NVARRAY_ADD Procedure Parameters

Parameter	Description
p_array	The name-value array instance. On input, the array to modify. If NULL, a new array is created. On output, the modified array.
p_value	The value to add. If NULL, p_array is not changed.

NVARRAY_GET Function

This function gets the name-value element of the name you specify in p_name from a name-value array.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN SYS.MGW_NAME_VALUE_T;
```

Parameters

Table 32–12 NVARAAY_GET Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The matching element, or NULL if the specified name is not found.

NVARRAY_GET_BOOLEAN Function

This function gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BOOLEAN (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 32–13 NVARRAY_GET_BOOLEAN Function Parameters

Parameter	Description
p_array	The name-value array

Table 32–13 NVARRAY_GET_BOOLEAN Function Parameters

Parameter	Description
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_BYTE Function

This function gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BYTE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 32–14 NVARRAY_GET_BYTE Function

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_SHORT Function

This function gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_SHORT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 32–15 NVARRAY_GET_SHORT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_INTEGER Function

This function gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_INTEGER (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

Parameters

Table 32–16 NVARRAY_GET_INTEGER Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_LONG Function

This function gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_LONG (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 32–17 NVARRAY_GET_LONG Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_FLOAT Function

This function gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_FLOAT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 32–18 NVARRAY_GET_FLOAT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_DOUBLE Function

This function gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_DOUBLE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 32–19 NVARRAY_GET_DOUBLE Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_TEXT Function

This function gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_TEXT (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN VARCHAR2;
```

Parameters

Table 32–20 NVARRAY_GET_TEXT Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_RAW Function

This function gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_RAW (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN RAW;
```

Parameters

Table 32–21 NVARRAY_GET_RAW Function Parameters

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_GET_DATE Function

This function gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_DATE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN DATE;
```

Parameters

Table 32–22 NVARRAY_GET_DATE Function Parameters

Parameters	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

The value, or NULL if the specified name is not found or if a type mismatch exists.

NVARRAY_FIND_NAME Function

This function searches a name-value array for the element with the name you specify in p_name.

Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

Parameters

Table 32–23 NVARRAY_FIND_NAME Function Parameters

Parameters	Description
p_array	The name-value array to search
p_name	The name to find
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

- A positive integer that is the array index of the matching element
- 0 if the specified name is not found

NVARRAY_FIND_NAME_TYPE Function

This function searches a name-value array for an element with the name and value type you specify.

Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME_TYPE (
   p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
   p_name     IN VARCHAR2,
   p_type     IN BINARY_INTEGER
   p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

Parameters

Table 32–24 NVARRAY_FIND_NAME_TYPE Function Parameters

Parameter	Description
p_array	The name-value array to search
p_name	The name to find
p_type	The value type. Refer to the value type constants in Table 32–7 on page 32-8.
p_compare	Name comparison method. Values: CASE_SENSITIVE, CASE_INSENSITIVE

Returns

- A positive integer that is the array index of the matching element
- 0 if the specified name is not found
- -1 if the specified name is found but a type mismatch exists

33

DBMS_MVIEW

DBMS_MVIEW enables you to understand capabilities for materialized views and potential materialized views, including their rewrite availability. It also enables you to refresh materialized views that are not part of the same refresh group and purge logs.

This chapter discusses the following topics:

- Summary of DBMS_MVIEW Subprograms

Note: DBMS_SNAPSHOT is a synonym for DBMS_MVIEW.

See Also:

- *Oracle9i Replication* for more information about using materialized views in a replication environment
- *Oracle9i Data Warehousing Guide* for more information about using materialized views in a data warehousing environment

Summary of DBMS_MVIEW Subprograms

Table 33–1 DBMS_MVIEW Package Subprograms

Subprogram	Description
BEGIN_TABLE_ REORGANIZATION Procedure on page 33-3	Performs a process to preserve materialized view data needed for refresh.
END_TABLE_ REORGANIZATION Procedure on page 33-4	Ensures that the materialized view data for the master table is valid and that the master table is in the proper state.
EXPLAIN_MVIEW Procedure on page 33-4	Explains what is possible with a materialized view or potential materialized view.
EXPLAIN_REWRITE Procedure on page 33-5	Explains why a query failed to rewrite.
I_AM_A_REFRESH Function on page 33-6	Returns the value of the `I_AM_REFRESH` package state.
PMARKER Function on page 33-7	Returns a partition marker from a rowid. This function is used for Partition Change Tracking (PCT).
PURGE_DIRECT_LOAD_LOG Procedure on page 33-7	Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing).
PURGE_LOG Procedure on page 33-7	Purges rows from the materialized view log.
PURGE_MVIEW_FROM_LOG Procedure on page 33-8	Purges rows from the materialized view log.
REFRESH Procedure on page 33-10	Refreshes one or more materialized views that are not members of the same refresh group.
REFRESH_ALL_MVIEWS Procedure on page 33-12	Refreshes all materialized views that do not reflect changes to their master table or master materialized view.
REFRESH_DEPENDENT Procedure on page 33-14	Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views.
REGISTER_MVIEW Procedure on page 33-16	Enables the administration of individual materialized views.

Table 33–1 DBMS_MVIEW Package Subprograms (Cont.)

Subprogram	Description
UNREGISTER_MVIEW Procedure on page 33-18	Enables the administration of individual materialized views. Invoked at a master site or master materialized view site to unregister a materialized view.

Note: If a query is less than 256 characters long, you can invoke `EXPLAIN_REWRITE()` using the `EXECUTE` command from SQL*PLUS. Otherwise, the recommended method is to use a PL/SQL `BEGIN..END` block, as shown in the examples in `/rdbms/demo/smxrw.sql`. The `EXPLAIN_REWRITE()` API cannot accept queries longer than 32627 characters. These restrictions also apply when passing the defining query of a materialized view to the `EXPLAIN_MVIEW` procedure.

BEGIN_TABLE_REORGANIZATION Procedure

This procedure performs a process to preserve materialized view data needed for refresh. It must be called before a master table is reorganized.

Syntax

```
DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2,
   tabname      IN   VARCHAR2);
```

Parameters

Table 33–2 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
`tabowner`	Owner of the table being reorganized.
`tabname`	Name of the table being reorganized.

END_TABLE_REORGANIZATION Procedure

This procedure ensures that the materialized view data for the master table is valid and that the master table is in the proper state. It must be called after a master table is reorganized.

Syntax

```
DBMS_MVIEW.END_TABLE_REORGANIZATION (
   tabowner    IN   VARCHAR2,
   tabname     IN   VARCHAR2);
```

Parameters

Table 33–3 END_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

EXPLAIN_MVIEW Procedure

This procedure enables you to learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

Using this procedure is straightforward. You simply call DBMS_MVIEW.EXPLAIN_MVIEW, passing in as parameters the schema and materialized view name for an existing materialized view. Alternatively, you can specify the SELECT string for a potential materialized view. The materialized view or potential materialized view is then analyzed and the results are written into either a table called MV_CAPABILITIES_TABLE, which is the default, or to an array called MSG_ARRAY.

Note that you must run the utlxmv.sql script prior to calling EXPLAIN_MVIEW except when you direct output to a VARRAY. The script is found in the admin directory. In addition, you must create MV_CAPABILITIES_TABLE in the current schema.

Syntax

The following PL/SQL declarations that are made for you in the DBMS_MVIEW package show the order and datatypes of these parameters for explaining an existing materialized view and a potential materialized view with output to a table and to a VARRAY.

To explain an existing or potential materialized view with output to MV_CAPABILITIES_TABLE:

```
DBMS_MVIEW.EXPLAIN_MVIEW (
 mv            IN VARCHAR2,
 statement_id  IN VARCHAR2:= NULL);
```

To explain an existing or potential materialized view with output to a VARRAY:

```
DBMS_MVIEW.EXPLAIN_MVIEW (
 mv          IN VARCHAR2,
 msg_array   OUT SYS.ExplainMVArrayType);
```

Parameters

Table 33–4 EXPLAIN_MVIEW Procedure Parameters

Parameter	Description
mv	The name of an existing materialized view (optionally qualified with the owner name separated by a ".") or a SELECT statement for a potential materialized view.
statement_id	A client-supplied unique identifier to associate output rows with specific invocations of EXPLAIN_MVIEW.
msg_array	The PL/SQL varray that receives the output. Use this parameter to direct EXPLAIN_MVIEW's output to a PL/SQL VARRAY rather than MV_CAPABILITIES_TABLE.

EXPLAIN_REWRITE Procedure

This procedure enables you to learn why a query failed to rewrite, or, if it rewrites, which materialized views will be used. Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the EXPLAIN_REWRITE statement is never actually executed.

To obtain the output into a table, you must run the admin/utlxrw.sql script before calling EXPLAIN_REWRITE. This script creates a table named REWRITE_TABLE in the current schema.

Syntax

You can obtain the output from EXPLAIN_REWRITE in two ways. The first is to use a table, while the second is to create a VARRAY. The following shows the basic syntax for using an output table:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           IN VARCHAR2,
    mv              IN VARCHAR2,
    statement_id    IN VARCHAR2;
```

If you want to direct the output of EXPLAIN_REWRITE to a varray, instead of a table, then the procedure should be called as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           IN VARCHAR2(2000),
    mv              IN VARCHAR2(30),
    msg_array       IN OUT SYS.RewriteArrayType);
```

Parameters

Table 33–5 EXPLAIN_REWRITE Procedure Parameters

Parameter	Description
query	SQL select statement to be explained.
mv	The fully qualified name of an existing materialized view in the form of SCHEMA.MV
statement_id	A client-supplied unique identifier to distinguish output messages
msg_array	The PL/SQL varray that receives the output. Use this parameter to direct EXPLAIN_REWRITE's output to a PL/SQL VARRAY

I_AM_A_REFRESH Function

This function returns the value of the I_AM_REFRESH package state. A return value of TRUE indicates that all local replication triggers for materialized views are effectively disabled in this session because each replication trigger first checks this state. A return value of FALSE indicates that these triggers are enabled.

Syntax

```
DBMS_MVIEW.I_AM_A_REFRESH()
  RETURN BOOLEAN;
```

PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

Syntax

```
DBMS_MVIEW.PMARKER(rid IN ROWID)
  RETURN NUMBER;
```

Parameters

Table 33–6 PMARKER Procedure Parameters

Parameter	Description
rid	The rowid of a row entry in a master table.

PURGE_DIRECT_LOAD_LOG Procedure

This procedure removes entries from the direct loader log after they are no longer needed for any known materialized view. This procedure usually is used in environments using Oracle's data warehousing technology.

See Also: *Oracle9i Data Warehousing Guide* for more information

Syntax

```
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG();
```

PURGE_LOG Procedure

This procedure purges rows from the materialized view log.

Syntax

```
DBMS_MVIEW.PURGE_LOG (
```

```
master          IN   VARCHAR2,
num             IN   BINARY_INTEGER := 1,
flag            IN   VARCHAR2       := 'NOP');
```

Parameters

Table 33–7 PURGE_LOG Procedure Parameters

Parameter	Description
master	Name of the master table or master materialized view.

num	Number of least recently refreshed materialized views whose rows you want to remove from materialized view log. For example, the following statement deletes rows needed to refresh the two least recently refreshed materialized views:
	`DBMS_MVIEW.PURGE_LOG('master_table', 2);`
	To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example:
	`DBMS_MVIEW.PURGE_LOG('master_table',9999);`
	This statement completely purges the materialized view log that corresponds to `master_table` if fewer than 9999 materialized views are based on `master_table`. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.

flag	Specify `delete` to guarantee that rows are deleted from the materialized view log for at least one materialized view. This parameter can override the setting for the parameter `num`. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view: `DBMS_MVIEW.PURGE_LOG('master_table',1,'delete');`

PURGE_MVIEW_FROM_LOG Procedure

This procedure is called on the master site or master materialized view site to delete the rows in materialized view refresh related data dictionary tables maintained at the master for the specified materialized view identified by its `mview_id` or the combination of the `mviewowner`, `mviewname`, and the `mviewsite`. If the materialized view specified is the oldest materialized view to have refreshed from any of the master tables or master materialized views, then the materialized view log is also purged. This procedure does not unregister the materialized view.

If there is an error while purging one of the materialized view logs, the successful purge operations of the previous materialized view logs are not rolled back. This is to minimize the size of the materialized view logs. In case of an error, this procedure can be invoked again until all the materialized view logs are purged.

Syntax

```
DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
   mview_id       IN   BINARY_INTEGER  |
   mviewowner     IN   VARCHAR2,
   mviewname      IN   VARCHAR2,
   mviewsite      IN   VARCHAR2);
```

> **Note:** This procedure is overloaded. The `mview_id` parameter is mutually exclusive with the three remaining parameters: `mviewowner`, `mviewname`, and `mviewsite`.

Parameters

Table 33–8 PURGE_MVIEW_FROM_LOG Procedure Parameters

Parameter	Description
`mview_id`	If you want to execute this procedure based on the identification of the target materialized view, specify the materialized view identification using the `mview_id` parameter. Query the `DBA_BASE_TABLE_MVIEWS` view at the materialized view log site for a listing of materialized view IDs. Executing this procedure based on the materialized view identification is useful if the target materialized view is not listed in the list of registered materialized views (`DBA_REGISTERED_MVIEWS`).
`mviewowner`	If you do not specify a `mview_id`, enter the owner of the target materialized view using the `mviewowner` parameter. Query the `DBA_REGISTERED_MVIEWS` view at the materialized view log site to view the materialized view owners.
`mviewname`	If you do not specify a `mview_id`, enter the name of the target materialized view using the `mviewname` parameter. Query the `DBA_REGISTERED_MVIEWS` view at the materialized view log site to view the materialized view names.

Table 33–8 PURGE_MVIEW_FROM_LOG Procedure Parameters

Parameter	Description
mviewsite	If you do not specify a mview_id, enter the site of the target materialized view using the mviewsite parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view sites.

REFRESH Procedure

This procedure refreshes a list of materialized views.

Syntax

```
DBMS_MVIEW.REFRESH (
   { list                IN     VARCHAR2,
   | tab                 IN OUT DBMS_UTILITY.UNCL_ARRAY,}
   method                IN     VARCHAR2       := NULL,
   rollback_seg          IN     VARCHAR2       := NULL,
   push_deferred_rpc     IN     BOOLEAN        := true,
   refresh_after_errors  IN     BOOLEAN        := false,
   purge_option          IN     BINARY_INTEGER := 1,
   parallelism           IN     BINARY_INTEGER := 0,
   heap_size             IN     BINARY_INTEGER := 0,
   atomic_refresh        IN     BOOLEAN        := true);
```

Note: This procedure is overloaded. The list and tab parameters are mutually exclusive.

Parameters

Table 33–9 REFRESH Procedure Parameters **(Page 1 of 2)**

Parameter	Description
list | tab	Comma-separated list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a materialized view.

method	A string of refresh methods indicating how to refresh the listed materialized views. An f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent.
	
	If a materialized view does not have a corresponding refresh method (that is, if more materialized views are specified than refresh methods), then that materialized view is refreshed according to its default refresh method. For example, consider the following EXECUTE statement within SQL*Plus:
	
	`DBMS_MVIEW.REFRESH`
	`('countries_mv,regions_mv,hr.employees_mv','cf');`
	
	This statement performs a complete refresh of the countries_mv materialized view, a fast refresh of the regions_mv materialized view, and a default refresh of the hr.employees materialized view.

rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views.
push_deferred_rpc	Used by updatable materialized views only. Set this parameter to true if you want to push changes from the materialized view to its associated master tables or master materialized views before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.

Table 33–9 REFRESH Procedure Parameters **(Page 2 of 2)**

Parameter	Description
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set purge to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set this parameter to 0 and occasionally execute PUSH with this parameter set to 2 to reduce the queue.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
atomic_refresh	If this parameter is set to true, then the list of materialized views is refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the materialized views is refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is false.

REFRESH_ALL_MVIEWS Procedure

This procedure refreshes all materialized views that have the following properties:

- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS (
   number_of_failures     OUT   BINARY_INTEGER,
   method                 IN    VARCHAR2         := NULL,
   rollback_seg           IN    VARCHAR2         := NULL,
   refresh_after_errors   IN    BOOLEAN          := false,
   atomic_refresh         IN    BOOLEAN          := true);
```

Parameters

Table 33–10 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
number_of_ failures	Returns the number of failures that occurred during processing.
method	A single refresh method indicating the type of refresh to perform for each materialized view that is refreshed. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. If no method is specified, a materialized view is refreshed according to its default refresh method.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views.
refresh_after_ errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_ refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
atomic_refresh	If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the refreshed materialized views is refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is false.

REFRESH_DEPENDENT Procedure

This procedure refreshes all materialized views that have the following properties:

- The materialized view depends on a master table or master materialized view in the list of specified masters.
- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.REFRESH_DEPENDENT (
   number_of_failures     OUT    BINARY_INTEGER,
   { list                 IN     VARCHAR2,
   | tab                  IN OUT DBMS_UTILITY.UNCL_ARRAY,}
   method                 IN     VARCHAR2    := NULL,
   rollback_seg           IN     VARCHAR2    := NULL,
   refresh_after_errors   IN     BOOLEAN     := false,
   atomic_refresh         IN     BOOLEAN     := true);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 33–11 REFRESH_DEPENDENT Procedure Parameters **(Page 1 of 3)**

Parameter	Description
number_of_ failures	Returns the number of failures that occurred during processing.

Table 33–11 REFRESH_DEPENDENT Procedure Parameters **(Page 2 of 3)**

Parameter	Description
list | tab	Comma-separated list of master tables or master materialized views on which materialized views can depend. (Synonyms are not supported.) These tables and the materialized views that depend on them can be located in different schemas. However, all of the tables and materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table.
method	A string of refresh methods indicating how to refresh the dependent materialized views. All of the materialized views that depend on a particular table are refreshed according to the refresh method associated with that table. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any materialized view that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `DBMS_MVIEW.REFRESH_DEPENDENT ('employees,deptartments,hr.regions','cf');` performs a complete refresh of the materialized views that depend on the employees table, a fast refresh of the materialized views that depend on the departments table, and a default refresh of the materialized views that depend on the hr.regions table.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views.
refresh_after_ errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_ refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.

Table 33–11 REFRESH_DEPENDENT Procedure Parameters **(Page 3 of 3)**

Parameter	Description
atomic_refresh	If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the refreshed materialized views is refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is false.

REGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to register a materialized view.

> **Note:** Typically, a materialized view is registered automatically during materialized view creation. You should only run this procedure to manually register a materialized view if the automatic registration failed or if the registration information was deleted.

Syntax

```
DBMS_MVIEW.REGISTER_MVIEW (
   mviewowner  IN   VARCHAR2,
   mviewname   IN   VARCHAR2,
   mviewsite   IN   VARCHAR2,
   mview_id    IN   DATE | BINARY_INTEGER,
   flag        IN   BINARY_INTEGER,
   qry_txt     IN   VARCHAR2,
   rep_type    IN   BINARY_INTEGER := DBMS_MVIEW.REG_UNKNOWN);
```

Parameters

Table 33–12 REGISTER_MVIEW Procedure Parameters

Parameter	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site for a materialized view registering at an Oracle8 and higher master site or master materialized view site. This name should not contain any double quotes.
mview_id	The identification number of the materialized view. Specify an Oracle8 and higher materialized view as a BINARY_INTEGER. Specify an Oracle7 materialized view registering at an Oracle8 and higher master sites or master materialized view sites as a DATE.
flag	A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following: ▪ dbms_mview.reg_rowid_mview for a rowid materialized view ▪ dbms_mview.reg_primary_key_mview for a primary key materialized view ▪ dbms_mview.reg_object_id_mview for an object id materialized view ▪ dbms_mview.reg_fast_refreshable_mview for a materialized view that can be fast refreshed ▪ dbms_mview.reg_updatable_mview for a materialized view that is updatable A materialized view can have more than one of these properties. In this case, use the plus sign (+) to specify more than one property. For example, if a primary key materialized view can be fast refreshed, you can enter the following for this parameter: dbms_mview.reg_primary_key_mview + dbms_mview.reg_fast_refreshable_mview You can determine the properties of a materialized view by querying the ALL_MVIEWS data dictionary view.
qry_txt	The first 32,000 bytes of the materialized view definition query.

Table 33–12 REGISTER_MVIEW Procedure Parameters

Parameter	Description
rep_type	Version of the materialized view. Valid constants that can be assigned include the following: ▪ dbms_mview.reg_v7_snapshot if the materialized view is at an Oracle7 site ▪ dbms_mview.reg_v8_snapshot if the materialized view is at an Oracle8 or higher site ▪ dbms_mview.reg_unknown (the default) if you do not know whether the materialized view is at an Oracle7 site or an Oracle8 (or higher) site

Usage Notes

This procedure is invoked at the master site or master materialized view site by a remote materialized view site using a remote procedure call. If REGISTER_MVIEW is called multiple times with the same mviewowner, mviewname, and mviewsite, then the most recent values for mview_id, flag, and qry_txt are stored. If a query exceeds the maximum VARCHAR2 size, then qry_txt contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the value of mview_id must be looked up in the materialized view data dictionary views by the person who calls the procedure.

UNREGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to unregister a materialized view.

Syntax

```
DBMS_MVIEW.UNREGISTER_MVIEW (
   mviewowner      IN   VARCHAR2,
   mviewname       IN   VARCHAR2,
   mviewsite       IN   VARCHAR2);
```

Parameters

Table 33–13 UNREGISTER_MVIEW Procedure Parameters

Parameters	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site.

34

DBMS_OBFUSCATION_TOOLKIT

DBMS_OBFUSCATION_TOOLKIT allows an application to encrypt data using either the Data Encryption Standard (DES) or the Triple DES algorithms.

The Data Encryption Standard (DES), also known as the Data Encryption Algorithm (DEA) by the American National Standards Institute (ANSI) and DEA-1 by the International Standards Organization (ISO), has been a worldwide encryption standard for over 20 years. The banking industry has also adopted DES-based standards for transactions between private financial institutions, and between financial institutions and private individuals. DES will eventually be replaced by a new Advanced Encryption Standard (AES).

DES is a symmetric key cipher; that is, the same key is used to encrypt data as well as decrypt data. DES encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm ignores 8 bits of the 64-bit key that is supplied; however, developers must supply a 64-bit key to the algorithm.

Triple DES (3DES) is a far stronger cipher than DES; the resulting ciphertext (encrypted data) is much harder to break using an exhaustive search: 2**112 or 2**168 attempts instead of 2**56 attempts. Triple DES is also not as vulnerable to certain types of cryptanalysis as is DES. DES procedures are as follows:

- DESEncrypt Procedure
- DESDecrypt Procedure

Oracle installs this package in the SYS schema. You can then grant package access to existing users and roles as needed. The package also grants access to the PUBLIC role so no explicit grant needs to be done.

This chapter discusses the following topics:

- Overview of Key Management
- Summary of DBMS_OBFUSCATION Subprograms

Overview of Key Management

Key management, including both generation and secure storage of cryptographic keys, is one of the most important aspects of encryption. If keys are poorly chosen or stored improperly, then it is far easier for a malefactor to break the encryption. Rather than using an exhaustive key search attack (that is, cycling through all the possible keys in hopes of finding the correct decryption key), cryptanalysts typically seek weaknesses in the choice of keys, or the way in which keys are stored.

Key generation is an important aspect of encryption. Typically, keys are generated automatically through a random-number generator. Provided that the random number generation is cryptographically secure, this can be an acceptable form of key generation. However, if random numbers are not cryptographically secure, but have elements of predictability, the security of the encryption may be easily compromised.

The `DBMS_OBFUSCATION_TOOLKIT` package does not generate encryption keys nor does it maintain them. Care must be taken by the application developer to ensure the secure generation and storage of encryption keys used with this package. Furthermore, the encryption and decryption done by the `DBMS_OBFUSCATION_TOOLKIT` takes place on the server, not the client. If the key is passed over the connection between the client and the server, the connection must be protected using Oracle Advanced Security; otherwise the key is vulnerable to capture over the wire.

Key storage is one of the most important, yet difficult aspects of encryption and one of the hardest to manage properly. To recover data encrypted with a symmetric key, the key must be accessible to the application or user seeking to decrypt data. The key needs to be easy enough to retrieve that users can access encrypted data when they need to without significant performance degradation. The key also needs to be secure enough that it is not easily recoverable by an unauthorized user trying to access encrypted data he is not supposed to see.

The three options available to a developer are:

- Store the key in the database
- Store the key in the operating system
- Have the user manage the key

Storing the Key in the Database

Storing the keys in the database cannot always provide bullet-proof security if you are trying to protect data against the DBA accessing encrypted data (since an all-privileged DBA can access tables containing encryption keys), but it can provide

security against the casual snooper, or against someone compromising the database files on the operating system. Furthermore, the security you can obtain by storing keys in the database does not have to be bullet-proof in order to be extremely useful.

For example, suppose you want to encrypt an employee's social security number, one of the columns in table EMP. You could encrypt each employee's SSN using a key which is stored in a separate column in EMP. However, anyone with SELECT access on the EMP table could retrieve the encryption key and decrypt the matching social security number. Alternatively, you could store the encryption keys in another table, and use a package to retrieve the correct key for the encrypted data item, based on a primary key-foreign key relationship between the tables.

A developer could envelope both the `DBMS_OBFUSCATION_TOOLKIT` package and the procedure to retrieve the encryption keys supplied to the package. Furthermore, the encryption key itself could be transformed in some way (for example, XORed with the foreign key to the EMP table) so that the key itself is not stored in easily recoverable form.

Oracle recommends using the wrap utility of PL/SQL to obfuscate the code within a PL/SQL package itself that does the encryption. That prevents people from breaking the encryption by looking at the PL/SQL code that handles keys, calls encrypting routines, and so on. In other words, use the wrap utility to obfuscate the PL/SQL packages themselves. This scheme is secure enough to prevent users with `SELECT` access to `EMP` from reading unencrypted sensitive data, and a DBA from easily retrieving encryption keys and using them to decrypt data in the `EMP` table. It can be made more secure by changing encryption keys regularly, or having a better key storage algorithm (so the keys themselves are encrypted, for example).

Storing the Key in the Operating System

Storing keys in the operating system (that is, in a flat file) is another option. With Oracle8*i* you can make callouts from PL/SQL, which you could use to retrieve encryption keys. If you store keys in the O/S and make callouts to retrieve the keys, the security of your encrypted data is only as secure as the protection of the key file on the O/S. Of course, a user retrieving keys from the operating system would have to be able to either access the Oracle database files (to decrypt encrypted data), or be able to gain access to the table in which the encrypted data is stored as a legitimate user.

User-Supplied Keys

If you ask a user to supply the key, it is crucial that you use network encryption, such as that provided by Oracle Advanced Security, so the key is not passed from

client to server in the clear. The user must remember the key, or your data is nonrecoverable.

Summary of DBMS_OBFUSCATION Subprograms

Table 34–1 DBMS_OBFUSCATION Subprograms

Subprogram	Description
DESEncrypt Procedure on page 34-4	Generates the encrypted form of the input data.
DESDecrypt Procedure on page 34-5	Generates the decrypted form of the input data.
DES3Encrypt Procedure on page 34-8	Generates the encrypted form of the input data by passing it through the Triple DES (3DES) encryption algorithm.
DES3Decrypt Procedure on page 34-10	Generates the decrypted form of the input data.

DESEncrypt Procedure

The DESEncrypt procedure generates the encrypted form of the input data. An example of the DESEncrypt procedure appears at the end of this chapter.

The DES algorithm encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm throws away 8 bits of the supplied key (the particular bits which are thrown away is beyond the scope of this documentation). However, developers using the algorithm must supply a 64-bit key or the package will raise an error.

Parameters

Table 34–2 DESEncrypt Parameters for Raw Data

Parameter Name	Mode	Type	Description
`input`	IN	RAW	data to be encrypted
`key`	IN	RAW	encryption key
`encrypted_data`	OUT	RAW	encrypted data

Table 34–3 DESEncrypt Parameters for String Data

Parameter Name	Mode	Type	Description
input_string	IN	VARCHAR2	string to be encrypted
key_string	IN	VARCHAR2	encryption key string
encrypted_string	OUT	VARCHAR2	encrypted string

If the input data or key given to the PL/SQL DESEncrypt procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit".

If the input data given to the DESEncrypt procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit".

If the user tries to double encrypt data using the DESEncrypt procedure, then the procedure raises the error ORA-28233 "Double encryption not supported".

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

Restrictions

The DESEncryption procedure has two restrictions. The first is that the DES key length for encryption is fixed at 56 bits; you cannot alter this key length.

The second is that you cannot execute multiple passes of encryption. That is, you cannot re-encrypt previously encrypted data by calling the function twice.

> **Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of US regulations governing the export of cryptographic products.

DESDecrypt Procedure

The purpose of the DESDecrypt procedure is to generate the decrypted form of the input data. An example of the DESDecrypt procedure appears at the end of this chapter.

Parameters

Table 34–4 DESDecrypt Parameters for Raw Data

Parameter Name	Mode	Type	Description
input	IN	RAW	Data to be decrypted
key	IN	RAW	Decryption key
decrypted_data	OUT	RAW	Decrypted data

Table 34–5 DESDecrypt Parameters for String Data

Parameter Name	Mode	Type	Description
input_string	IN	VARCHAR2	String to be decrypted
key_string	IN	VARCHAR2	Decryption key string
decrypted_string	OUT	VARCHAR2	Decrypted string

If the input data or key given to the PL/SQL DESDecrypt function is empty, then Oracle raises ORA error 28231 "Invalid input to Obfuscation toolkit".

If the input data given to the DESDecrypt function is not a multiple of 8 bytes, Oracle raises ORA error 28232 "Invalid input size for Obfuscation toolkit".

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

Note: ORA-28233 is not applicable to the DESDecrypt function.

Restrictions

The DES key length for encryption is fixed at 64 bits (of which 56 bits are used); you cannot alter this key length.

Note: The key length limitation is a requirement of U.S. regulations governing the export of cryptographic products.

Example

A sample PL/SQL program follows. Segments of the code are numbered and contain narrative text explaining portions of the code.

```
DECLARE
   input_string        VARCHAR2(16) := 'tigertigertigert';
   raw_input           RAW(128) := UTL_RAW.CAST_TO_RAW(input_string);
   key_string          VARCHAR2(8)  := 'scottsco';
   raw_key             RAW(128) := UTL_RAW.CAST_TO_RAW(key_string);
   encrypted_raw               RAW(2048);
   encrypted_string            VARCHAR2(2048);
   decrypted_raw               RAW(2048);
   decrypted_string            VARCHAR2(2048);
   error_in_input_buffer_length EXCEPTION;
   PRAGMA EXCEPTION_INIT(error_in_input_buffer_length, -28232);
   INPUT_BUFFER_LENGTH_ERR_MSG VARCHAR2(100) :=
    '*** DES INPUT BUFFER NOT A MULTIPLE OF 8 BYTES - IGNORING
EXCEPTION ***';
   double_encrypt_not_permitted EXCEPTION;
   PRAGMA EXCEPTION_INIT(double_encrypt_not_permitted, -28233);
   DOUBLE_ENCRYPTION_ERR_MSG VARCHAR2(100) :=
    '*** CANNOT DOUBLE ENCRYPT DATA - IGNORING EXCEPTION ***';

-- 1. Begin testing raw data encryption and decryption
   BEGIN
   dbms_output.put_line('> ========= BEGIN TEST RAW DATA =========');
   dbms_output.put_line('> Raw input                          : ' ||
                UTL_RAW.CAST_TO_VARCHAR2(raw_input));
   BEGIN
      dbms_obfuscation_toolkit.DESEncrypt(input => raw_input,
               key => raw_key, encrypted_data => encrypted_raw );
      dbms_output.put_line('> encrypted hex value              : ' ||
               rawtohex(encrypted_raw));
      dbms_obfuscation_toolkit.DESDecrypt(input => encrypted_raw,
               key => raw_key, decrypted_data => decrypted_raw);
      dbms_output.put_line('> Decrypted raw output             : ' ||
                    UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw));
      dbms_output.put_line('>  ');
      if UTL_RAW.CAST_TO_VARCHAR2(raw_input) =
                    UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw) THEN
         dbms_output.put_line('> Raw DES Encyption and Decryption successful');
      END if;
   EXCEPTION
      WHEN error_in_input_buffer_length THEN
             dbms_output.put_line('> ' || INPUT_BUFFER_LENGTH_ERR_MSG);
```

```
   END;
   dbms_output.put_line('>  ');


-- 2. Begin testing string data encryption and decryption
   dbms_output.put_line('> ========= BEGIN TEST STRING DATA =========');

   BEGIN
      dbms_output.put_line('> input string                    : '
                           || input_string);
      dbms_obfuscation_toolkit.DESEncrypt(
               input_string => input_string,
               key_string => key_string,
               encrypted_string => encrypted_string );
      dbms_output.put_line('> encrypted hex value             : ' ||
                  rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_string)));
      dbms_obfuscation_toolkit.DESDecrypt(
               input_string => encrypted_string,
               key_string => key_string,
               decrypted_string => decrypted_string );
      dbms_output.put_line('> decrypted string output         : ' ||
                 decrypted_string);
      if input_string = decrypted_string THEN
         dbms_output.put_line('> String DES Encyption and Decryption
successful');
      END if;
   EXCEPTION
      WHEN error_in_input_buffer_length THEN
            dbms_output.put_line(' ' || INPUT_BUFFER_LENGTH_ERR_MSG);
   END;
   dbms_output.put_line('>  ');
END;
```

DES3Encrypt Procedure

The DES3Encrypt procedure generates the encrypted form of the input data by passing it through the Triple DES (3DES) encryption algorithm. An example of the DESEncrypt procedure appears at the end of this chapter.

Oracle's implementation of 3DES supports either a 2-key or 3-key implementation, in outer cipher-block-chaining (CBC) mode.

A developer using Oracle's 3DES interface with a 2-key implementation must supply a single key of 128 bits as an argument to the DES3Encrypt procedure. With

a 3-key implementation, you must supply a single key of 192 bits. Oracle then breaks the supplied key into two 64-bit keys. As with DES, the 3DES algorithm throws away 8 bits of each derived key. However, you must supply a single 128-bit key for the 2-key 3DES implementation or a single 192-bit key for the 3-key 3DES implementation; otherwise the package will raise an error. The DES3Encrypt procedure uses the 2-key implementation by default.

Parameters

Table 34–6 DES3Encrypt Parameters for Raw Data

Parameter Name	Mode	Type	Description
input	IN	RAW	data to be encrypted
key	IN	RAW	encryption key
encrypted_data	OUT	RAW	encrypted data
which	IN	PLS_INTEGER	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.

Table 34–7 DES3Encrypt Parameters for String Data

Parameter Name	Mode	Type	Description
input_string	IN	VARCHAR2	string to be encrypted
key_string	IN	VARCHAR2	encryption key string
encrypted_string	OUT	VARCHAR2	encrypted string
which	IN	PLS_INTEGER	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.

If the input data or key given to the PL/SQL DES3Encrypt procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit".

If the input data given to the DES3Encrypt procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit".

If the user tries to double encrypt data using the DES3Encrypt procedure, then the procedure raises the error ORA-28233 "Double encryption not supported".

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the WHICH parameter, ORA-28236 "Invalid Triple DES mode" is generated. Only the values 0 (TwoKeyMode) and 1 (ThreeKeyMode) are valid.

Restrictions

The DES3Encrypt procedure has two restrictions. The first is that the DES key length for encryption is fixed at 128 bits (for 2-key DES) or 192 bits (for 3-key DES); you cannot alter these key lengths.

The second is that you cannot execute multiple passes of encryption using 3DES. (Note: the 3DES algorithm itself encrypts data multiple times; however, you cannot call the 3DESencrypt function itself more than once to encrypt the same data using 3DES.)

Note: Both the key length limitation and the prevention of multiple encryption passes are requirements of US regulations governing the export of cryptographic products.

DES3Decrypt Procedure

The purpose of the DES3Decrypt procedure is to generate the decrypted form of the input data. An example of the DES3Decrypt procedure appears at the end of this chapter.

Parameters

Table 34–8 DES3Decrypt Parameters for Raw Data

Parameter Name	Mode	Type	Description
`input`	IN	RAW	Data to be decrypted
`key`	IN	RAW	Decryption key
`decrypted_data`	OUT	RAW	Decrypted data

Table 34–8 DES3Decrypt Parameters for Raw Data

Parameter Name	Mode	Type	Description
which	IN	PLS_INTEGER	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.

Table 34–9 DES3Decrypt parameters for string data

Parameter Name	Mode	Type	Description
input_string	IN	VARCHAR2	String to be decrypted
key_string	IN	VARCHAR2	Decryption key string
decrypted_ string	OUT	VARCHAR2	Decrypted string
which	IN	PLS_INTEGER	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.

If the input data or key given to the DES3Decrypt procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit".

If the input data given to the DES3Decrypt procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit". ORA-28233 is NOT applicable for the DES3Decrypt function.

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the WHICH parameter, ORA-28236 "Invalid Triple DES mode" is generated. Only the values 0 (TwoKeyMode) and 1 (ThreeKeyMode) are valid.

Restrictions

A developer must supply a single key of either 128 bits for a 2-key implementation (of which only 112 are used), or a single key of 192 bits for a 3-key implementation (of which 168 bits are used). Oracle automatically truncates the supplied key into 56-bit lengths for decryption. This key length is fixed and cannot be altered.

Note: Both the key length limitation and the prevention of multiple encryption passes are requirements of US regulations governing the export of cryptographic products.

Example

Following is a sample PL/SQL program for your reference. Segments of the code are numbered and contain narrative text explaining portions of the code.

```
DECLARE
   input_string        VARCHAR2(16) := 'tigertigertigert';
   raw_input           RAW(128) := UTL_RAW.CAST_TO_RAW(input_string);
   key_string          VARCHAR2(16)  := 'scottscottscotts';
   raw_key             RAW(128) := UTL_RAW.CAST_TO_RAW(key_string);
encrypted_raw               RAW(2048);
   encrypted_string            VARCHAR2(2048);
decrypted_raw               RAW(2048);
   decrypted_string            VARCHAR2(2048);
   error_in_input_buffer_length EXCEPTION;
   PRAGMA EXCEPTION_INIT(error_in_input_buffer_length, -28232);
   INPUT_BUFFER_LENGTH_ERR_MSG VARCHAR2(100) :=
    '*** DES INPUT BUFFER NOT A MULTIPLE OF 8 BYTES - IGNORING EXCEPTION ***';
   double_encrypt_not_permitted EXCEPTION;
   PRAGMA EXCEPTION_INIT(double_encrypt_not_permitted, -28233);
   DOUBLE_ENCRYPTION_ERR_MSG VARCHAR2(100) :=
    '*** CANNOT DOUBLE ENCRYPT DATA - IGNORING EXCEPTION ***';

-- 1. Begin testing raw data encryption and decryption
   BEGIN
   dbms_output.put_line('> ========= BEGIN TEST RAW DATA =========');
   dbms_output.put_line('> Raw input                          : ' ||
                UTL_RAW.CAST_TO_VARCHAR2(raw_input));
   BEGIN
      dbms_obfuscation_toolkit.DES3Encrypt(input => raw_input,
               key => raw_key, encrypted_data => encrypted_raw );
      dbms_output.put_line('> encrypted hex value              : ' ||
               rawtohex(encrypted_raw));
      dbms_obfuscation_toolkit.DES3Decrypt(input => encrypted_raw,
               key => raw_key, decrypted_data => decrypted_raw);
      dbms_output.put_line('> Decrypted raw output             : ' ||
                   UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw));
      dbms_output.put_line('>  ');
```

```
      if UTL_RAW.CAST_TO_VARCHAR2(raw_input) =
                   UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw) THEN
         dbms_output.put_line('> Raw DES3 Encyption and Decryption successful');
      END if;
   EXCEPTION
      WHEN error_in_input_buffer_length THEN
            dbms_output.put_line('> ' || INPUT_BUFFER_LENGTH_ERR_MSG);
   END;
   dbms_output.put_line('>  ');
END;

-- 2. Begin testing string data encryption and decryption
   dbms_output.put_line('> ========= BEGIN TEST STRING DATA =========');

   BEGIN
      dbms_output.put_line('> input string                    : '
                           || input_string);
      dbms_obfuscation_toolkit.DES3Encrypt(
               input_string => input_string,
               key_string => key_string,
               encrypted_string => encrypted_string );
      dbms_output.put_line('> encrypted hex value             : ' ||
                  rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_string)));
      dbms_obfuscation_toolkit.DES3Decrypt(
               input_string => encrypted_string,
               key_string => key_string,
               decrypted_string => decrypted_string );
      dbms_output.put_line('> decrypted string output         : ' ||
                 decrypted_string);
      if input_string = decrypted_string THEN
         dbms_output.put_line('> String DES3 Encyption and Decryption
successful');
      END if;
   EXCEPTION
      WHEN error_in_input_buffer_length THEN
            dbms_output.put_line(' ' || INPUT_BUFFER_LENGTH_ERR_MSG);
   END;
   dbms_output.put_line('>  ');
END;
```

35

DBMS_ODCI

DBMS_ODCI returns the CPU cost of a user function based on the elapsed time of the function. The CPU cost is used by extensible optimizer routines.

This chapter discusses the following topics:

- Summary of DBMS_ODCI Subprograms

Summary of DBMS_ODCI Subprograms

Table 35–1 DBMS_ODCI Subprograms

Subprogram	Description
ESTIMATE_CPU_UNITS Function on page 35-2	Returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds).

ESTIMATE_CPU_UNITS Function

ESTIMATE_CPU_UNITS returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds). This information can be used to associate the CPU cost with a user-defined function for the extensible optimizer.

The function takes as input the elapsed time of the user function, measures CPU units by multiplying the elapsed time by the processor speed of the machine, and returns the approximate number of CPU instructions that should be associated with the user function. (For a multiprocessor machine, ESTIMATE_CPU_UNITS considers the speed of a single processor.)

Syntax

```
DBMS_ODCI.ESTIMATE_CPU_UNITS (
   elapsed_time NUMBER)
RETURN NUMBER;
```

Parameters

Table 35–2 ESTIMATE_CPU_UNITS Function Parameters

Parameter	Description
elapsed_time	The elapsed time in seconds to execute the function

Usage Notes

When associating CPU cost with a user-defined function, use the full number of CPU units rather than the number of *thousands* of CPU units returned by ESTIMATE_CPU_UNITS. In other words, multiply the number returned by ESTIMATE_CPU_UNITS by 1000.

Example

To determine the number of CPU units used for a function that takes 10 seconds on a machine:

```
DECLARE
  a INTEGER;
BEGIN
  a := DBMS_ODCI.ESTIMATE_CPU_UNITS(10);
  DBMS_OUTPUT.PUT_LINE('CPU units = '|| a*1000);
END;
```

36

DBMS_OFFLINE_OG

The DBMS_OFFLINE_OG package contains public APIs for offline instantiation of master groups.

This chapter discusses the following topics:

- Summary of DBMS_OFFLINE_OG Subprograms

Note: These procedures are used in performing an offline instantiation of a master table in a multimaster replication environment.

These procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of DBMS_OFFLINE_OG Subprograms

Table 36–1 DBMS_OFFLINE_OG Package Subprograms

Subprogram	Description
BEGIN_INSTANTIATION Procedure on page 36-2	Starts offline instantiation of a master group.
BEGIN_LOAD Procedure on page 36-3	Disables triggers while data is imported to new master site as part of offline instantiation.
END_INSTANTIATION Procedure on page 36-5	Completes offline instantiation of a master group.
END_LOAD Procedure on page 36-6	Re-enables triggers after importing data to new master site as part of offline instantiation.
RESUME_SUBSET_OF_MASTERS Procedure on page 36-7	Resumes replication activity at all existing sites except the new site during offline instantiation of a master group.

BEGIN_INSTANTIATION Procedure

This procedure starts offline instantiation of a master group. You must call this procedure from the master definition site.

Note: This procedure is used to perform an offline instantiation of a master table in a multimaster replication environment.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 36–2 BEGIN_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the replication group that you want to replicate to the new site.
new_site	The fully qualified database name of the new site to which you want to replicate the replication group.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 36–3 BEGIN_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
dbms_repcat.nonmasterdef	This procedure must be called from the master definition site.
sitealreadyexists	Specified site is already a master site for this replication group.
wrongstate	Status of master definition site must be quiesced.
dbms_ repcat.missingrepgroup	gname does not exist as a master group.
dbms_repcat.missing_ flavor	If you receive this exception, contact Oracle Support Services.

BEGIN_LOAD Procedure

This procedure disables triggers while data is imported to the new master site as part of offline instantiation. You must call this procedure from the new master site.

Note: This procedure is used to perform an offline instantiation of a master table in a multimaster replication environment.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 36–4 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group whose members you are importing.
new_site	The fully qualified database name of the new site at which you will be importing the replication group members.

Exceptions

Table 36–5 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Specified site is not recognized by replication group.
wrongstate	Status of the new master site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a master group.

END_INSTANTIATION Procedure

This procedure completes offline instantiation of a master group. You must call this procedure from the master definition site.

> **Note:** This procedure is used to perform an offline instantiation of a master table in a multimaster replication environment.
>
> This procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.END_INSTANTIATION (
  gname     IN  VARCHAR2,
  new_site  IN  VARCHAR2);
```

Parameters

Table 36–6 END_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the replication group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the replication group.

Exceptions

Table 36–7 END_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
dbms_repcat.nonmasterdef	This procedure must be called from the master definition site.
unknownsite	Specified site is not recognized by replication group.
wrongstate	Status of master definition site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a master group.

END_LOAD Procedure

This procedure re-enables triggers after importing data to new master site as part of offline instantiation. You must call this procedure from the new master site.

Note: This procedure is used to perform an offline instantiation of a master table in a multimaster replication environment.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.END_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 36–8 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group whose members you have finished importing.
new_site	The fully qualified database name of the new site at which you have imported the replication group members.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 36–9 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Specified site is not recognized by replication group.
wrongstate	Status of the new master site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a master group.
dbms_repcat.flavor_noobject	If you receive this exception, contact Oracle Support Services.
dbms_repcat.flavor_contains	If you receive this exception, contact Oracle Support Services.

RESUME_SUBSET_OF_MASTERS Procedure

When you add a new master site to a master group by performing an offline instantiation of a master site, it may take some time to complete the offline instantiation process. This procedure resumes replication activity at all existing sites, except the new site, during offline instantiation of a master group. You typically execute this procedure after executing the DBMS_OFFLINE_OG.BEGIN_

INSTANTIATION procedure. You must call this procedure from the master definition site.

> **Note:** This procedure is used to perform an offline instantiation of a master table in a multimaster replication environment.
>
> This procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2
   override  IN  BOOLEAN := false);
```

Parameters

Table 36–10 RESUME_SUBSET_OF_MASTERS Procedure Parameters

Parameter	Description
gname	Name of the replication group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the replication group.
override	If this is true, then any pending RepCat administrative requests are ignored and normal replication activity is restored at each master as quickly as possible. The override parameter should be set to true only in emergency situations. If this is false, then normal replication activity is restored at each master only when there is no pending RepCat administrative request for gname at that master.

Exceptions

Table 36–11 RESUME_SUBSET_OF_MASTERS Procedure Exceptions

Exception	Description
`badargument`	`NULL` or empty string for replication group or new master site name.
`dbms_repcat.nonmasterdef`	This procedure must be called from the master definition site.
`unknownsite`	Specified site is not recognized by replication group.
`wrongstate`	Status of master definition site must be quiesced.
`dbms_repcat.missingrepgroup`	`gname` does not exist as a master group.

37

DBMS_OFFLINE_SNAPSHOT

The DBMS_OFFLINE_SNAPSHOT package contains public APIs for offline instantiation of materialized views.

This chapter discusses the following topics:

- Summary of DBMS_OFFLINE_SNAPSHOT Subprograms

> **Note:** These procedure are used in performing an offline instantiation of a materialized view.
>
> These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of DBMS_OFFLINE_SNAPSHOT Subprograms

Table 37–1 DBMS_OFFLINE_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_LOAD Procedure on page 37-2	Prepares a materialized view site for import of a new materialized view as part of offline instantiation.
END_LOAD Procedure on page 37-4	Completes offline instantiation of a materialized view.

BEGIN_LOAD Procedure

This procedure prepares a materialized view site for import of a new materialized view as part of offline instantiation. You must call this procedure from the materialized view site for the new materialized view.

Note: This procedure is used to perform an offline instantiation of a materialized view.

These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
   gname               IN   VARCHAR2,
   sname               IN   VARCHAR2,
   master_site         IN   VARCHAR2,
   snapshot_oname      IN   VARCHAR2,
   storage_c           IN   VARCHAR2 := '',
   comment             IN   VARCHAR2 := '',
   min_communication   IN   BOOLEAN  := true);
```

Parameters

Table 37–2 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group for the materialized view that you are creating using offline instantiation.
sname	Name of the schema for the new materialized view.
master_site	Fully qualified database name of the materialized view's master site.
snapshot_oname	Name of the temporary materialized view created at the master site.
storage_c	Storage options to use when creating the new materialized view at the materialized view site.
comment	User comment.
min_communication	If true, then the update trigger sends the new value of a column only if the update statement modifies the column. Also, if true, the update trigger sends the old value of the column only if it is a key column or a column in a modified column group.

Exceptions

Table 37–3 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group, schema, master site, or materialized view name.
dbms_ repcat.missingrepgroup	gname does not exist as a replication group.
missingremotemview	Could not locate specified materialized view at specified master site.
dbms_ repcat.missingschema	Specified schema does not exist.
mviewtabmismatch	Base table name of the materialized view at the master and materialized view do not match.

END_LOAD Procedure

This procedure completes offline instantiation of a materialized view. You must call this procedure from the materialized view site for the new materialized view.

Note: This procedure is used to perform an offline instantiation of a materialized view.

These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
   gname           IN  VARCHAR2,
   sname           IN  VARCHAR2,
   snapshot_oname  IN  VARCHAR2);
```

Parameters

Table 37–4 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group for the materialized view that you are creating using offline instantiation.
sname	Name of the schema for the new materialized view.
snapshot_oname	Name of the materialized view.

Exceptions

Table 37–5 END_LOAD Procedure Exceptions

Exception	Description
`badargument`	`NULL` or empty string for replication group, schema, or materialized view name.
`dbms_repcat.missingrepgroup`	`gname` does not exist as a replication group.
`dbms_repcat.nonmview`	This procedure must be called from the materialized view site.

38

DBMS_OLAP

The DBMS_OLAP package provides a collection of materialized view analysis and advisory functions that are callable from any PL/SQL program. Some of the functions generate output tables.

> **See Also:** *Oracle9i Data Warehousing Guide* for more information regarding how to use DBMS_OLAP and its output tables

This chapter discusses the following topics:

- Requirements
- Error Messages
- Summary of DBMS_OLAP Subprograms
- DBMS_OLAP Interface Views

Requirements

DBMS_OLAP performs seven major functions, which include materialized view strategy recommendation, materialized view strategy evaluation, reporting and script generation, repository management, workload management, filter management, and dimension validation.

To perform materialized view strategy recommendation and evaluation functions, the workload information can either be provided by the user or synthesized by the Advisor engine. In the former case, cardinality information of all tables and materialized views referenced in the workload are required. In the latter case, dimension objects must be present and cardinality information for all dimension tables, fact tables, and materialized views are required. Cardinality information should be gathered with the DBMS_STATS.GATHER_TABLE_STATS procedure. Once these functions are completed, the analysis results can be presented with the reporting and script generation function.

The workload management function handles three types of workload, which are user-specified workload, SQL cache workload, and Oracle Trace workload. To process the user-specified workload, a user-defined workload table must be present in the user's schema. To process Oracle Trace workload, Oracle Trace formatter must be run to preprocess collected workload statistics into default V-tables in the user's schema.

Error Messages

Table 38–1 lists basic DBMS_OLAP error messages.

Table 38–1 DBMS_OLAP Error Messages

Error Code	Description
ORA-30442	Cannot find the definition for filter <NUMBER>
ORA-30443	Definition for filter <NUMBER>'s item <NUMBER> is invalid
ORA-30444	Rewrite terminated by the SQL Analyzer
ORA-30445	Workload queries not found
ORA-30446	Valid workload queries not found
ORA-30447	internal data for run number <NUMBER> is inconsistent
ORA-30448	Internal data of the Advisor repository is inconsistent
ORA-30449	Syntax error in parameter <NUMBER>

Table 38–1 DBMS_OLAP Error Messages

Error Code	Description
ORA-30465	Supplied run_id is not valid: <NUMBER>
ORA-30466	Cannot find the specified workload <NUMBER>
ORA-30477	The input select_clause is incorrectly specified
ORA-30478	Specified dimension does not exist
ORA-30479	Summary Advisor error QSM message with more details
QSM-00501	Unable to initialize Summary Advisor environment
QSM-00502	OCI error
QSM-00503	Out of memory
QSM-00504	Internal error
QSM-00505	Syntax error in <parse_entity_name> - <error_description>
QSM-00506	No fact-tables could be found
QSM-00507	No dimensions could be found
QSM-00508	Statistics missing on tables / columns
QSM-00509	Invalid parameter - <parameter_name>
QSM-00510	Statistics missing on summaries
QSM-00511	Invalid fact-tables specified in fact-filter
QSM-00512	Invalid summaries specified in the retention-list
QSM-00513	One or more of the workload tables is missing
QSM-00550	The filter item type <NAME> is missing the required data
QSM-00551	The file <NAME> was not found
QSM-00552	The workload source was not defined or was not recognized
QSM-00553	The string value for filter item <NAME> has a maximum length of <NUMBER> characters
QSM-00554	The required table name was not provided
QSM-00555	The table <NAME> cannot be accessed or does not exist
QSM-00556	The file <NAME> could not be opened
QSM-00557	The owner <NAME> cannot be accessed or does not exist

Table 38–1 DBMS_OLAP Error Messages

Error Code	Description
QSM-00558	An error occurred while reading file <NAME>
QSM-00559	A workload already exists for the specified collection ID
QSM-00560	The character <NAME> is invalid at line <LINE_NUMBER>, column <COLUMN_NUMBER
QSM-00561	Found <TOKEN> at line <NUMBER>, column <NUMBER>. Expecting 1of the following items: <ITEMS>
QSM-00562	The requested Advisor task was not found
QSM-00563	Found <TOKEN> at line <NUMBER>, column <NUMBER> of file <NAME>. Expecting 1 of the following items: <ITEMS>
QSM-00564	An internal lexical error occurred: <Additional error text>
QSM-00565	The <NAME> was not found while validating the <TABLE or COLUMN> at line <NUMBER>, column <NUMBER>
QSM-00566	The <TOKEN> is ambiguous while validating the <TABLE or COLUMN> at line <NUMBER>, column <NUMBER>
QSM-00567	A runtime error occurred: <Additional error text>
QSM-00568	The end-of-file was encountered
QSM-00569	The required column <NAME> was not found in table <NAME>
QSM-00570	The job has ended in error. Status changes are not permitted
QSM-00571	The job has already completed. Status changes are unnecessary
QSM-00572	No repository connection has been established
QSM-00573	The date <VALUE> must be in the form 'DD/MM/YYYY HH24:MI:SS'
QSM-00574	The file <NAME> could not be accessed due to a security violation
QSM-00575	The string <VALUE> cannot be converted to a number
QSM-00576	A usable Oracle Trace collection was not found in schema <NAME>
QSM-00577	The current operation was cancelled by the user
QSM-00578	A temporary file cannot be created using the specification <FILE_NAME>
QSM-00579	The job has already completed. Cancellation is unnecessary
QSM-00580	The job has ended in error. Cancellation is not permitted
QSM-00581	Internal error: <Additional error text>

Table 38–1 DBMS_OLAP Error Messages

Error Code	Description
QSM-00582	A database error has occurred. <Additional error text>
QSM-00583	The filter item type <NAME> is invalid
QSM-00584	The SQL cache is not accessible by user <NAME>
QSM-00585	The workload was not found for collection ID <NUMBER>
QSM-00586	The filter was not found for filter ID <NUMBER>
QSM-00587	The analysis data was not found for run ID <NUMBER>
QSM-00588	The current user does not have the privilege to access the requested workload, which is owned by user <NAME>
QSM-00589	The current user does not have the privilege to access the requested workload filter, which is owned by user <NAME>
QSM-00590	The current user does not have the privilege to access the requested Advisor items, which are owned by user <NAME>
QSM-00591	The specified report style <NAME> was not found
QSM-00592	The specified report field <NAME> already exists
QSM-00593	The specified report field <NAME> was not found
QSM-00594	The specified ID number is already being used by another user
QSM-00595	The specified ID number is being used by an Advisor <NAME> object and cannot be used for this operation
QSM-00596	A specified ID number cannot be NULL or zero
QSM-00597	Found <TOKEN> at line <NUMBER>, column <NUMBER>
QSM-00598	The minimum range value for filter item <NAME> is greater than the maximum range value
QSM-00599	The supplied workload filter contains items that are unsupported for the requested workload operation: <OPERATION>
QSM-00602	The ID <NUMBER> is not a valid Summary Advisor run or collection ID for the current user
QSM-00601	The flags value of <NUMBER> for the Summary Advisor detail report is invalid

Summary of DBMS_OLAP Subprograms

Table 38–2 lists the subprograms available with DBMS_OLAP.

Table 38–2 DBMS_OLAP Package Subprograms

Subprogram	Description
ADD_FILTER_ITEM Procedure on page 38-7	Filters the contents being used during the recommendation process.
CREATE_ID Procedure on page 38-9	Generates an internal ID used by a new workload collection, a new filter, or a new advisor run
ESTIMATE_MVIEW_SIZE Procedure on page 38-9	Estimates the size of a materialized view that you might create, in bytes and rows.
EVALUATE_MVIEW_ STRATEGY Procedure on page 38-10	Measures the utilization of each existing materialized view.
GENERATE_MVIEW_REPORT Procedure on page 38-11	Generates an HTML-based report on the given Advisor run
GENERATE_MVIEW_SCRIPT Procedure on page 38-12	Generates a simple script containing the SQL commands to implement Summary Advisor recommendations
LOAD_WORKLOAD_CACHE Procedure on page 38-13	Obtains a SQL cache workload.
LOAD_WORKLOAD_TRACE Procedure on page 38-14	Loads a workload collected by Oracle Trace.
LOAD_WORKLOAD_USER Procedure on page 38-15	Loads a user-defined workload.
PURGE_FILTER Procedure on page 38-16	Deletes a specific filter or all filters.
PURGE_RESULTS Procedure on page 38-17	Removes all results or those for a specific run.
PURGE_WORKLOAD Procedure on page 38-17	Deletes all workloads or a specific collection.
RECOMMEND_MVIEW_ STRATEGY Procedure on page 38-18	Generates a set of recommendations about which materialized views should be created, retained, or dropped.
SET_CANCELLED Procedure on page 38-19	Stops the Advisor if it takes too long returning results.

Table 38–2 DBMS_OLAP Package Subprograms (Cont.)

Subprogram	Description
VALIDATE_DIMENSION Procedure on page 38-20	Verifies that the relationships specified in a `dimension` are correct.
VALIDATE_WORKLOAD_CACHE Procedure on page 38-21	Validates the SQL Cache workload before performing load operations
VALIDATE_WORKLOAD_TRACE Procedure on page 38-22	Validates the Oracle Trace workload before performing load operations
VALIDATE_WORKLOAD_USER Procedure on page 38-22	Validates the user-supplied workload before performing load operations

ADD_FILTER_ITEM Procedure

This procedure adds a new filter item to an existing filter to make it more restrictive. It also creates a filter to restrict what is analyzed for the workload.

Syntax

```
ADD_FILTER_ITEM (
   filter_id    IN NUMBER,
   filter_name  IN VARCHAR2,
   string_list  IN VARCHAR2,
   number_min   IN NUMBER,
   number_max   IN NUMBER,
   date_min     IN VARCHAR2,
   date_max     IN VARCHAR2);
```

Table 38–3 ADD_FILTER_ITEM Procedure Parameters

Parameter	Datatype	Description
`filter_id`	`NUMBER`	An ID that uniquely describes the filter. It is generated by the `DBMS_OLAP.CREATE_ID` procedure

Table 38–3 ADD_FILTER_ITEM Procedure Parameters

Parameter	Datatype	Description
filter_name	VARCHAR2	APPLICATION String-workload's application column. An example of how to load a SQL Cache workload follows: BASETABLE String-based tables referenced by workload queries. Name must be fully qualified including owner and table name (for example, SH.SALES) CARDINALITY Numerical-sum of cardinality of the referenced base tables FREQUENCY Numerical-workload's frequency column LASTUSE Date-workload's lastuse column. Not used by SQL Cache workload. OWNER String-workload's owner column. Expected in uppercase unless owner defined explicitly to be not all in uppercase. PRIORITY Numerical-workload's priority column. Not used by SQL Cache workload. RESPONSETIME Numerical-workload's responsetime column. Not used by SQL Cache workload. SCHEMA String-based schema referenced by workload filter. TRACENAME String-list of oracle trace collection names. Only used by a Trace Workload
string_list	VARCHAR2	A comma-delimited list of strings. This parameter is only used by the filter items of the string type
number_min	NUMBER	The lower bound of a numerical range. NULL represents the lowest possible value. This parameter is only used by the parameters of the numerical type
number_max	NUMBER	The upper bound of a numerical range, NULL for no upper bound. NULL represents the highest possible value. This parameter is only used by the parameters of the numerical type

Table 38–3 ADD_FILTER_ITEM Procedure Parameters

Parameter	Datatype	Description
date_min	VARCHAR2	The lower bound of a date range. NULL represents the lowest possible date value. This parameter is only used by the parameters of the date type
date_max	VARCHAR2	The upper bound of a date range. NULL represents the highest possible date value. This parameter is only used by the parameters of the date type

CREATE_ID Procedure

This creates a unique identifier, which is used to identify a filter, a workload or results of an advisor or dimension validation run.

Syntax

```
CALL DBMS_OLAP.CREATE_ID (
  id        OUT NUMBER);
```

Table 38–4 CREATE_ID Procedure Parameters

Parameter	Datatype	Description
id	NUMBER	The unique identifier that can be used to identify a filter, a workload, or an Advisor run

ESTIMATE_MVIEW_SIZE Procedure

This estimates the size of a materialized view that you might create, in bytes and number of rows.

Syntax

```
DBMS_OLAP.ESTIMATE_MVIEW_SIZE (
   stmt_id       IN  VARCHAR2,
   select_clause IN  VARCHAR2,
   num_rows      OUT NUMBER,
   num_bytes     OUT NUMBER);
```

Parameters

Table 38–5 ESTIMATE_MVIEW_SIZE Procedure Parameters

Parameter	Datatype	Description
stmt_id	NUMBER	Arbitrary string used to identify the statement in an EXPLAIN PLAN.
select_clause	STRING	The SELECT statement to be analyzed.
num_rows	NUMBER	Estimated cardinality.
num_bytes	NUMBER	Estimated number of bytes.

EVALUATE_MVIEW_STRATEGY Procedure

This procedure measures the utilization of each existing materialized view based on the materialized view usage statistics collected from the workload. The workload_id is optional. If not provided, EVALUATE_MVIEW_STRATEGY uses a hypothetical workload.

Periodically, the unused results can be purged from the system by calling the DBMS_OLAP.PURGE_RESULTS procedure.

See Also: "DBMS_OLAP Interface Views" on page 38-23

Syntax

```
DBMS_OLAP.EVALUATE_MVIEW_STRATEGY (
run_id        IN NUMBER,
workload_id   IN NUMBER,
filter_id     IN NUMBER);
```

Parameters

Table 38–6 EVALUATE_MVIEW_STRATEGY Procedure Parameters

Parameter	Datatype	Description
run_id	NUMBER	An ID generated by the DBMS_OLAP.CREATE_ID procedure to identify results of a run

Table 38–6 EVALUATE_MVIEW_STRATEGY Procedure Parameters

Parameter	Datatype	Description
workload_id	NUMBER	An optional workload ID that maps to a workload in the current repository. Use the parameter DBMS_OLAP.WORKLOAD_ALL to choose all workloads
filter_id	NUMBER	Specify filter for the workload to be used. The value DBMS_OLAP.FILTER_NONE indicates no filtering

GENERATE_MVIEW_REPORT Procedure

Generates an HTML-based report on the given Advisor run.

Syntax

```
DBMS_OLAP.GENERATE_MVIEW_REPORT (
filename     IN VARCHAR2,
id           IN NUMBER,
flags        IN NUMBER);
```

Table 38–7 GENERATE_MVIEW_REPORT Procedure Parameters

Parameter	Datatype	Description
filename	VARCHAR2	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permissions
id	NUMBER	An ID that identifies an advisor run. Or use the parameter DBMS_OLAP.RUNID_ALL to indicate all advisor runs should be reported

Table 38–7 GENERATE_MVIEW_REPORT Procedure Parameters

Parameter	Datatype	Description
flags	NUMBER	Bit masked flags indicating what sections should be reported DBMS_OLAP.RPT_ACTIVITY -- Overall activities DBMS_OLAP.RPT_JOURNAL -- Runtime journals DBMS_OLAP.RPT_WORKLOAD_FILTER -- Filters DBMS_OLAP.RPT_WORKLOAD_DETAIL -- Workload information DBMS_OLAP.RPT_WORKLOAD_QUERY -- Workload query information DBMS_OLAP.RPT_RECOMMENDATION -- Recommendations DBMS_OLAP.RPT_USAGE -- Materialized view usage DBMS_OLAP.RPT_ALL -- All sections

GENERATE_MVIEW_SCRIPT Procedure

Generates a simple script containing the SQL commands to implement Summary Advisor recommendations.

Syntax

```
DBMS_OLAP.GENERATE_MVIEW_SCRIPT(
filename     IN VARCHAR2,
id           IN NUMBER,
tspace        IN VARCHAR2);
```

Table 38–8 GENERATE_MVIEW_SCRIPT Procedure Parameters

Parameter	Datatype	Description
filename	VARCHAR2	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permissions
id	NUMBER	An ID that identifies an advisor run. The parameter DBMS_OLAP.RUNID_ALL indicates all advisor runs should be reported.

Table 38–8 GENERATE_MVIEW_SCRIPT Procedure Parameters

Parameter	Datatype	Description
tspace	VARCHAR2	Optional tablespace name to use when creating materialized views.

LOAD_WORKLOAD_CACHE Procedure

Loads a SQL cache workload.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_CACHE (
workload_id  IN NUMBER,
flags        IN NUMBER,
filter_id    IN NUMBER,
application  IN VARCHAR2,
priority     IN NUMBER);
```

Table 38–9 LOAD_WORKLOAD_CACHE Procedure Parameters

Parameter	Datatype	Description
workload_id	NUMBER	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permission
flags	NUMBER	DBMS_OLAP.WORKLOAD_OVERWRITE The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID DBMS_OLAP.WORKLOAD_APPEND The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload DBMS_OLAP.WORKLOAD_NEW The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation

Table 38–9 LOAD_WORKLOAD_CACHE Procedure Parameters

Parameter	Datatype	Description
filter_id	NUMBER	Specify filter for the workload to be loaded
application	VARCHAR2	The default business application name. This value will be used for a query if one is not found in the target workload
priority	NUMBER	The default business priority to be assigned to every query in the target workload

LOAD_WORKLOAD_TRACE Procedure

Loads an Oracle Trace workload.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_TRACE (
workload_id  IN NUMBER,
flags        IN NUMBER,
filter_id    IN NUMBER,
application  IN VARCHAR2,
priority     IN NUMBER,
owner_name   IN VARCHAR2);
```

Table 38–10 LOAD_WORKLOAD_TRACE Procedure Parameters

Parameter	Datatype	Description
collectionid	NUMBER	Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the "Security and Performance" section of the Java Developer's Guide for more information on file permission

Table 38–10 LOAD_WORKLOAD_TRACE Procedure Parameters

Parameter	Datatype	Description
flags	NUMBER	DBMS_OLAP.WORKLOAD_OVERWRITE The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID DBMS_OLAP.WORKLOAD_APPEND The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload DBMS_OLAP.WORKLOAD_NEW The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation
filter_id	NUMBER	Specify filter for the workload to be loaded
application	VARCHAR2	The default business application name. This value will be used for a query if one is not found in the target workload
priority	NUMBER	The default business priority to be assigned to every query in the target workload
owner_name	VARCHAR2	The schema that contains the Oracle Trace data. If omitted, the current user will be used

LOAD_WORKLOAD_USER Procedure

A user-defined workload is loaded using the procedure LOAD_WORKLOAD_USER.

Syntax

```
DBMS_OLAP.LOAD_WORKLOAD_USER (
workload_id  IN NUMBER,
flags        IN NUMBER,
filter_id    IN NUMBER,
owner_name   IN VARCHAR2,
table_name   IN VARCHAR2);
```

Table 38–11 LOAD_WORKLOAD_USER Procedure Parameters

Parameter	Datatype	Description
workload_id	NUMBER	The required id that was returned by the DBMS_OLAP.CREATE_ID call
flags	NUMBER	DBMS_OLAP.WORKLOAD_OVERWRITE The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID DBMS_OLAP.WORKLOAD_APPEND The load routine preserves any existing queries in the workload. Any queries collected by the load operation will be appended to the end of the specified workload DBMS_OLAP.WORKLOAD_NEW The load routine assumes there are no existing queries in the workload. If it finds an existing workload element, the call will fail with an error Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation
filter_id	NUMBER	Specify filter for the workload to be loaded
owner_name	VARCHAR2	The schema that contains the user supplied table or view
table_name	VARCHAR2	The table or view name containing valid workload data

PURGE_FILTER Procedure

A filter can be removed at anytime by calling the procedure PURGE_FILTER which is described as follows. You can delete a specific filter or all filters.

Syntax

```
DBMS_OLAP.PURGE_FILTER (
filter_id    IN NUMBER);
```

Table 38–12 PURGE_FILTER Procedure Parameters

Parameter	Datatype	Description
filter_id	NUMBER	The parameter DBMS_OLAP.FILTER_ALL indicates all filters should be removed.

PURGE_RESULTS Procedure

Many procedures in the DBMS_OLAP package generate output in system tables, such as recommendation results for DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY and evaluation results for DBMS_OLAP.EVALUATE_MVIEW_STRATEGY, dimension validation results for DBMS_OLAP.VALIDATE_DIMENSION. These results can be accessed through a set of interface views, as shown in "DBMS_OLAP Interface Views" on page 38-23. When they are no longer required, they should be removed using the procedure PURGE_RESULTS. You can remove all results or those for a specific run.

Syntax

```
DBMS_OLAP.PURGE_RESULTS (
run_id IN NUMBER);
```

Parameters

Table 38–13 PURGE_RESULTS Procedure Parameters

Parameter	Datatype	Description
run_id	NUMBER	An ID generated with the DBMS_OLAP.CREATE_ID procedure. The ID should be associated with a DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY or a DBMS_OLAP.EVALUATE_MVIEW_STRATEGY or a DBMS_OLAP.VALIDATE_DIMENSION run. Use the value DBMS_OLAP.RUNID_ALL to specify all such runs

PURGE_WORKLOAD Procedure

When workloads are no longer needed, they can be removed using the procedure PURGE_WORKLOAD. You can delete all workloads or a specific collection.

Syntax

```
DBMS_OLAP.PURGE_WORKLOAD (
workload_id IN NUMBER);
```

Table 38–14 DBMS_OLAP.PURGE_WORKLOAD Procedure Parameters

Parameter	Datatype	Description
workload_id	NUMBER	An ID number originally assigned by the create_id call. If the value of workload_id is set to DBMS_OLAP.WORKLOAD_ALL, then all workloads for the current user will be deleted

RECOMMEND_MVIEW_STRATEGY Procedure

This procedure generates a set of recommendations about which materialized views should be created, retained, or dropped, based on information in the workload (gathered by Oracle Trace, the user workload, or the SQL cache), and an analysis of table and column cardinality statistics gathered by the DBMS_STATS.GATHER_TABLE_STATS procedure.

RECOMMEND_MVIEW_STRATEGY requires that you have run the DBMS_STATS.GATHER_TABLE_STATS procedure to gather table and column cardinality statistics and have collected and formatted the workload statistics.

The workload is aggregated to determine the count of each request in the workload, and this count is used as a weighting factor during the optimization process. If the workload_id is not provided, then RECOMMEND_MVIEW_STRATEGY uses a hypothetical workload based on dimension definitions and other embedded statistics.

The space of all dimensional materialized views that include the specified fact tables identifies the set of materialized views that optimize performance across the workload. The recommendation results are stored in system tables, which can be accessed through the view SYSTEM.MVIEW_RECOMMENDATIONS.

Periodically, the unused results can be purged from the system by calling the DBMS_OLAP.PURGE_RESULTS procedure

See Also: "DBMS_OLAP Interface Views" on page 38-23

Syntax

```
DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY (
   run_id              IN  NUMBER,
   workload_id         IN  NUMBER,
   filter_id           IN  NUMBER,
   storage_in_bytes    IN  NUMBER,
```

```
   retention_pct      IN  NUMBER,
   retention_list     IN  VARCHAR2,
   fact_table_filter  IN  VARCHAR2);
```

Parameters

Table 38–15 RECOMMEND_MVIEW_STRATEGY Procedure Parameters

Parameter	Description
run_id	An ID generated by the DBMS_OLAP.CREATE_ID procedure to uniquely identify results of a run
workload_id	An optional workload ID that maps to a workload in the current repository. Use the parameter DBMS_OLAP.WORKLOAD_ALL to choose all workloads. If the workload_id is set to NULL, the call will use a hypothetical workload
filter_id	An optional filter ID that maps to a set of user-supplied filter items. Use the parameter DBMS_OLAP.FILTER_NONE to avoid filtering
storage_in_bytes	Maximum storage, in bytes, that can be used for storing materialized views. This number must be nonnegative.
retention_pct	Number between 0 and 100 that specifies the percent of existing materialized view storage that must be retained, based on utilization on the actual or hypothetical workload. A materialized view is retained if the cumulative space, ranked by utilization, is within the retention threshold specified (or if it is explicitly listed in retention_list). Materialized views that have a NULL utilization (for example, nondimensional materialized views) are always retained.
retention_list	Comma-delimited list of materialized view table names. A drop recommendation is not made for any materialized view that appears in this list.
fact_table_filter	Optional list of fact tables used to filter real or ideal workload

SET_CANCELLED Procedure

If the Summary Advisor takes too long to make its recommendations using the procedures RECOMMEND_MVIEW_STRATEGY, you can stop it by calling the

procedure SET_CANCELLED and passing in the run_id for this recommendation process.

Syntax

```
DBMS_OLAP.SET_CANCELLED (
   run_id     IN  NUMBER);
```

Table 38–16 DBMS_OLAP.SET_CANCELLED Procedure Parameters

Parameter	Datatype	Description
run_id	NUMBER	Id that uniquely identifies an advisor analysis operation. This call can be used to cancel a long running workload collection as well as an Advisor analysis session

VALIDATE_DIMENSION Procedure

This procedure verifies that the hierarchical and attribute relationships, and join relationships, specified in an existing dimension object are correct. This provides a fast way to ensure that referential integrity is maintained.

The validation results are stored in system tables, which can be accessed through the view SYSTEM.MVIEW_EXCEPTIONS.

Periodically, the unused results can be purged from the system by calling the DBMS_OLAP.PURGE_RESULTS procedure.

See Also: "DBMS_OLAP Interface Views" on page 38-23

Syntax

```
DBMS_OLAP.VALIDATE_DIMENSION (
   dimension_name    IN VARCHAR2,
   dimension_owner   IN VARCHAR2,
   incremental       IN BOOLEAN,
   check_nulls       IN BOOLEAN,
   run_id            IN NUMBER);
```

Parameters

Table 38–17 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_name	Name of the dimension to analyze.
dimension_owner	Name of the dimension owner.
incremental	If TRUE, then tests are performed only for the rows specified in the sumdelta$ table for tables of this dimension; otherwise, check all rows.
check_nulls	If TRUE, then all level columns are verified to be nonnull; otherwise, this check is omitted. Specify FALSE when nonnullness is guaranteed by other means, such as NOT NULL constraints.
run_id	An ID generated by the DBMS_OLAP.CREATE_ID procedure to identify a run

VALIDATE_WORKLOAD_CACHE Procedure

This procedure validates the SQL Cache workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_CACHE (
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 38–18 VALIDATE_WORKLOAD_USER Procedure Parameters

Parameter	Description
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.
error	VARCHAR2, return error set

VALIDATE_WORKLOAD_TRACE Procedure

This procedure validates the Oracle Trace workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_TRACE (
   owner_name       IN  VARCHAR2,
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 38–19 VALIDATE_WORKLOAD_TRACE Procedure Parameters

Parameter	Description
owner_name	Owner of the trace workload table
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.
error	VARCHAR2, return error text

VALIDATE_WORKLOAD_USER Procedure

This procedure validates the user-supplied workload before performing load operations.

Syntax

```
DBMS_OLAP.VALIDATE_WORKLOAD_USER (
   owner_name       IN  VARCHAR2,
   table_name       IN  VARCHAR2,
   valid            OUT NUMBER,
   error            OUT VARCHAR2);
```

Parameters

Table 38–20 VALIDATE_WORKLOAD_USER Procedure Parameters

Parameter	Description
owner_name	Owner of the user workload table
table_name	User workload table name
valid	Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.
error	VARCHAR2, return error set

DBMS_OLAP Interface Views

Several views are created when using DBMS_OLAP. All are in the SYSTEM schema. To access these views, you must have a DBA role.

> **See Also:** *Oracle9i Data Warehousing Guide* for more information regarding how to use DBMS_OLAP

SYSTEM.MVIEW_EVALUATIONS

Table 38–21 SYSTEM.MVIEW_EVALUATIONS

Column	NULL?	Datatype	Description
RUNID	NOT NULL	NUMBER	Run id identifying a unique advisor call
MVIEW_OWNER	-	VARCHAR2(30)	Owner of materialized view
MVIEW_NAME	-	VARCHAR2(30)	Name of an exiting materialized view in this database
RANK	NOT NULL	NUMBER	Rank of this materialized view in descending order of benefit_to_cost_ ratio
STORAGE_IN_BYTES	-	NUMBER	Size of the materialized view in bytes

Table 38–21 SYSTEM.MVIEW_EVALUATIONS

Column	NULL?	Datatype	Description
FREQUENCY	-	NUMBER	Number of times this materialized view appears in the workload
CUMULATIVE_BENEFIT	-	NUMBER	The cumulative benefit of the materialized view
BENEFIT_TO_COST_RATIO	NOT NULL	NUMBER	The ratio of cumulative_benefit to storage_in_bytes

SYSTEM.MVIEW_EXCEPTIONS

Table 38–22 SYSTEM.MVIEW_EXCEPTIONS

Column	NULL?	Datatype	Description
RUNID	-	NUMBER	Run id identifying a unique advisor call
OWNER	-	VARCHAR2(30)	Owner name
TABLE_NAME	-	VARCHAR2(30)	Table name
DIMENSION_NAME	-	VARCHAR2(30)	Dimension name
RELATIONSHIP	-	VARCHAR2(11)	Violated relation name
BAD_ROWID	-	ROWID	Location of offending entry

SYSTEM.MVIEW_FILTER

Table 38–23 SYSTEM.MVIEW_FILTER

Column	NULL?	Datatype	Description
FILTERID	NOT NULL	NUMBER	Unique number used to identify the operation that used this filter
SUBFILTERNUM	NOT NULL	NUMBER	A unique id number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table
SUBFILTERTYPE	-	VARCHAR2(12)	Filter item number

Table 38–23 SYSTEM.MVIEW_FILTER

Column	NULL?	Datatype	Description
STR_VALUE	-	VARCHAR2(1028)	String attribute for items that require strings
NUM_VALUE1	-	NUMBER	Numeric low for items that require numbers
NUM_VALUE2	-	NUMBER	Numeric high for items that require numbers
DATE_VALUE1	-	DATE	Date low for items that require dates
DATE_VALUE2	-	DATE	Date high for items that require dates

SYSTEM.MVIEW_FILTERINSTANCE

Table 38–24 SYSTEM.MVIEW_FILTER

Column	NULL?	Datatype	Description
RUNID	NOT NULL	NUMBER	Unique number used to identify the operation that used this filter
FILTERID	-	NUMBER	A unique id number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table
SUBFILTERNUM	-	NUMBER	Filter item number
SUBFILTERTYPE	-	VARCHAR2(12)	Filter item type
STR_VALUE	-	VARCHAR2(1028)	String attribute for items that require strings
NUM_VALUE1	-	NUMBER	Numeric low for items that require numbers
NUM_VALUE2	-	NUMBER	Numeric high for items that require numbers
DATE_VALUE1	-	DATE	Date low for items that require dates
DATE_VALUE2	-	DATE	Date high for items that require dates

SYSTEM.MVIEW_LOG

Table 38–25 SYSTEM.MVIEW_LOG

Column	NULL?	Datatype	Description
ID	NOT NULL	NUMBER	Unique number used to identify the table entry. The number must be created using the CREATE_ID routine
FILTERID	-	NUMBER	Optional filter id. Zero indicates no user-supplied filter has been applied to the operation
RUN_BEGIN	-	DATE	Date at which the operation began
RUN_END	-	DATE	Date at which the operation ended
TYPE	-	VARCHAR2(11)	A name that identifies the type of operation
STATUS	-	VARCHAR2(11)	The current operational status
MESSAGE	-	VARCHAR2(2000)	Informational message indicating current operation or condition
COMPLETED	-	NUMBER	Number of steps completed by operation
TOTAL	-	NUMBER	Total number steps to be performed
ERROR_CODE	-	VARCHAR2(20)	Oracle error code in the event of an error

SYSTEM.MVIEW_RECOMMENDATIONS

Table 38–26 SYSTEM.MVIEW_RECOMMENDATIONS

Column	NULL?	Datatype	Description
RUNID	-	NUMBER	Run id identifying a unique advisor call
ALL_TABLES	-	VARCHAR2(2000)	A comma-delimited list of fully qualified table names for structured recommendations
FACT_TABLES	-	VARCHAR2(1000)	A comma-delimited list of grouping levels, if any, for structured recommendation
GROUPING_LEVELS	-	VARCHAR2(2000)	-

Table 38–26 SYSTEM.MVIEW_RECOMMENDATIONS

Column	NULL?	Datatype	Description
QUERY_TEXT	-	LONG	Query text of materialized view if RECOMMENDED_ACTION is CREATE; null otherwise
RECOMMENDATION_ NUMBER	NOT NULL	NUMBER	Unique identifier for this recommendation
RECOMMENDED_ACTION	-	VARCHAR2(6)	CREATE, RETAIN, or DROP, Retain, Create, or Drop
MVIEW_OWNER	-	VARCHAR2(30)	Owner of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise
MVIEW_NAME	-	VARCHAR2(30)	Name of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise
STORAGE_IN_BYTES	-	NUMBER	Actual or estimated storage in bytes
PCT_PERFORMANCE_GAIN	-	NUMBER	The expected incremental improvement in performance obtained by accepting this recommendation relative to the initial condition, assuming that all previous recommendations have been accepted, or NULL if unknown
BENEFIT_TO_COST_ RATIO	NOT NULL	NUMBER	Ratio of the incremental improvement in performance to the size of the materialized view in bytes, or NULL if unknown

SYSTEM.MVIEW_WORKLOAD

Table 38–27 SYSTEM.MVIEW_WORKLOAD

Column	NULL?	Datatype	Description
APPLICATION	-	VARCHAR2(30)	Optional application name for the query
CARDINALITY	-	NUMBER	Total cardinality of all of tables in query
WORKLOADID	-	NUMBER	Workload id identifying a unique sampling
FREQUENCY	-	NUMBER	Number of times query executed

Table 38–27 SYSTEM.MVIEW_WORKLOAD

Column	NULL?	Datatype	Description
IMPORT_TIME	-	DATE	Date at which item was collected
LASTUSE	-	DATE	Last date of execution
OWNER	-	VARCHAR2(30)	User who last executed query
PRIORITY	-	NUMBER	User-supplied ranking of query
QUERY	-	LONG	Query text
QUERYID	-	NUMBER	Id number identifying a unique query
RESPONSETIME	-	NUMBER	Execution time in seconds
RESULTSIZE	-	NUMBER	Total bytes selected by the query

39

DBMS_ORACLE_TRACE_AGENT

The `DBMS_ORACLE_TRACE_AGENT` package provides some system level utilities.

This chapter discusses the following topics:

- Security
- Summary of DBMS_ORACLE_TRACE_AGENT Subprograms

Security

This package is only accessible to user SYS by default. You can control access to these routines by only granting execute to privileged users.

Note: This package should only be granted to DBA or the Oracle TRACE collection agent.

Summary of DBMS_ORACLE_TRACE_AGENT Subprograms

This package contains only one subprogram: SET_ORACLE_TRACE_IN_SESSION.

SET_ORACLE_TRACE_IN_SESSION Procedure

This procedure collects Oracle Trace data for a database session other than your own. It enables Oracle TRACE in the session identified by (sid, serial#). These value are taken from v$session.

Syntax

```
DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION (
   sid                NUMBER   DEFAULT 0,
   serial#            NUMBER   DEFAULT 0,
   on_off IN          BOOLEAN  DEFAULT false,
   collection_name IN VARCHAR2 DEFAULT '',
   facility_name   IN VARCHAR2 DEFAULT '');
```

Parameters

Table 39–1 SET_ORACLE_TRACE_IN_SESSION Procedure Parameters

Parameter	Description
sid	Session ID.
serial#	Session serial number.
on_off	TRUE or FALSE. Turns tracing on or off.
collection_name	The Oracle TRACE collection name to be used.
facility_name	The Oracle TRACE facility name to be used.

Usage Notes

If the collection does not occur, then check the following:

- Be sure that the server event set file identified by <facility_name> exists. If there is no full file specification on this field, then the file should be located in the directory identified by ORACLE_TRACE_FACILITY_PATH in the initialization file.
- The following files should exist in your Oracle Trace admin directory: REGID.DAT, PROCESS.DAT, and COLLECT.DAT. If they do not, then you must run the OTRCCREF executable to create them.

  Note: PROCESS.DAT was changed to FACILITY.DAT with Oracle8.

- The stored procedure packages should exist in the database. If the packages do not exist, then run the OTRCSVR.SQL file (in your Oracle Trace or RDBMS admin directories) to create the packages.
- The user has the EXECUTE privilege on the stored procedure.

Example

```
EXECUTE DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION
(8,12,TRUE,'NEWCOLL','oracled');
```

40

DBMS_ORACLE_TRACE_USER

DBMS_ORACLE_TRACE_USER provides public access to the Oracle TRACE instrumentation for the calling user. Using the Oracle Trace stored procedures, you can invoke an Oracle Trace collection for your own session or for another session.

This chapter discusses the following topics:

Summary of DBMS_ORACLE_TRACE_USER Subprograms

This package contains only one subprogram: SET_ORACLE_TRACE.

SET_ORACLE_TRACE Procedure

This procedure collects Oracle Trace data for your own database session.

Syntax

```
DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE (
   on_off          IN BOOLEAN  DEFAULT false,
   collection_name IN VARCHAR2 DEFAULT '',
   facility_name   IN VARCHAR2 DEFAULT '');
```

Parameters

Table 40–1 SET_ORACLE_TRACE Procedure Parameters

Parameter	Description
on_off	TRUE or FALSE: Turns tracing on or off.
collection_name	Oracle TRACE collection name to be used.
facility_name	Oracle TRACE facility name to be used.

Example

```
EXECUTE DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE
(TRUE,'MYCOLL','oracle');
```

41

DBMS_OUTLN

The DBMS_OUTLN package, synonymous with OUTLN_PKG, contains the functional interface for subprograms associated with the management of stored outlines.

A stored outline is the stored data that pertains to an execution plan for a given SQL statement. It enables the optimizer to repeatedly re-create execution plans that are equivalent to the plan originally generated along with the outline.The data stored in an outline consists, in part, of a set of hints that are used to achieve plan stability.

This chapter discusses the following topics:

- Requirements and Security for DBMS_OUTLN
- Summary of DBMS_OUTLN Subprograms

Requirements and Security for DBMS_OUTLN

Requirements

DBMS_OUTLN contains management procedures that should be available to appropriate users only. EXECUTE privilege is not extended to the general user community unless the DBA explicitly does so.

Security

PL/SQL functions that are available for outline management purposes can be executed only by users with EXECUTE privilege on the procedure (or package).

Summary of DBMS_OUTLN Subprograms

Table 41–1 DBMS_OUTLN Package Subprograms

Subprogram	Description
DROP_BY_CAT Procedure on page 41-1	Drops outlines that belong to a specified category.
DROP_COLLISION Procedure on page 41-3	Drops an outline with an `ol$.hintcount` value that does not match the number of hints for that outline in `ol$hints`.
DROP_EXTRAS Procedure on page 41-4	Cleans up after an import by dropping extra hint tuples not accounted for by hintcount.
DROP_UNREFD_HINTS Procedure on page 41-4	Drops hint tuples that have no corresponding outline in the OL$ table.
DROP_BY_CAT Procedure on page 41-2	Drops outlines that have never been applied in the compilation of a SQL statement.
UPDATE_BY_CAT Procedure on page 41-5	Changes the category of outlines in one category to a new category.
GENERATE_SIGNATURE Procedure on page 41-6	Generates a signature for the specified SQL text.

DROP_BY_CAT Procedure

This procedure drops outlines that belong to a specified category.

Syntax

```
DBMS_OUTLN.DROP_BY_CAT
   cat VARCHAR2);
```

Parameters

Table 41–2 DROP_BY_CAT Procedure Parameters

Parameter	Description
cat	Category of outlines to drop.

Usage Notes

This procedure purges a category of outlines in a single call.

Example

This example drops all outlines in the DEFAULT category:

```
DBMS_OUTLN.DROP_BY_CAT('DEFAULT');
```

DROP_COLLISION Procedure

This procedure drops an outline with an ol$.hintcount value that does not match the number of hints for that outline in ol$hints.

Syntax

```
DBMS_OUTLN.DROP_COLLISION;
```

Usage Notes

A concurrency problem can occur if an outline is created or altered at the same time it is being imported. Because the outline must be imported according to its original design, if the concurrent operation changes the outline in mid-import, the outline will be dropped as unreliable based on the inconsistent metadata.

DROP_EXTRAS Procedure

This procedure cleans up after an import by dropping extra hint tuples not accounted for by hintcount.

Syntax

```
DBMS_OUTLN.DROP_EXTRAS;
```

Usage Notes

The OL$-tuple of an outline will be rejected if an outline already exists in the target database, either with the same name or the same signature. Hint tuples will also be rejected, up to the number of hints in the already existing outline. Therefore, if the rejected outline has more hint tuples than the existing one, spurious tuples will be inserted into the OL$HINTS table. This procedure, executed automatically as a post table action, will remove the wrongly inserted hint tuples.

DROP_UNREFD_HINTS Procedure

This procedure drops hint tuples that have no corresponding outline in the OLSable.

Syntax

```
DBMS_OUTLN.DROP_UNREFD_HINTS;
```

Usage Notes

This procedure will execute automatically as a post table action to remove hints with no corresponding entry in the OL$ table, a condition that can arise if an outline is dropped and imported concurrently.

DROP_UNUSED Procedure

This procedure drops outlines that have never been applied in the compilation of a SQL statement.

Syntax

```
DBMS_OUTLN.DROP_UNUSED;
```

Usage Notes

You can use DROP_UNUSED for outlines generated by an application for one-time use only, created as a result of dynamic SQL statements. These outlines are never used and take up valuable disk space.

UPDATE_BY_CAT Procedure

This procedure changes the category of all outlines in one category to a new category. If the SQL text in an outline already has an outline in the target category, it is not merged into the new category.

Syntax

```
DBMS.OUTLN.UPDATE_BY_CAT (
   oldcat VARCHAR2 DEFAULT 'DEFAULT',
   newcat VARCHAR2 DEFAULT 'DEFAULT');
```

Parameters

Table 41–3 UPDATE_BY_CAT Procedure Parameters

Parameter	Description
oldcat	Current category to be changed.
newcat	Target category to change outline to.

Usage Notes

Once satisfied with a set of outlines, you can move outlines from an *experimental* category to a *production* category. Likewise, you may want to merge a set of outlines from one category into another pre-existing category.

Example

This example changes all outlines in the DEFAULT category to the CAT1 category:

```
DBMS_OUTLN.UPDATE_BY_CAT('DEFAULT', 'CAT1');
```

GENERATE_SIGNATURE Procedure

This procedure generates a signature for the specified SQL text.

Syntax

```
DBMS_OUTLN.GENERATE_SIGNATURE (
   sqltxt      IN   VARCHAR2,
   signature   OUT  RAW);
```

Parameters

Table 41–4 GENERATE_SIGNATURE Procedure Parameters

Parameter	Description
sqltxt	The specified SQL.
signature	The signature to be generated.

42

DBMS_OUTLN_EDIT

The DBMS_OUTLN_EDIT package is an invoker's rights package.

This chapter discusses the following topics:

- Summary of DBMS_OUTLN_EDIT Subprograms

Summary of DBMS_OUTLN_EDIT Subprograms

Table 42–1 DBMS_OUTLN_EDIT Package Subprograms

Subprogram	Description
CHANGE_JOIN_POS Procedure on page 42-2	Changes the join position for the hint identified by outline name and hint number to the position specified by newpos.
CREATE_EDIT_TABLES Procedure on page 42-3	Creates outline editing tables in calling a user's schema.
DROP_EDIT_TABLES Procedure on page 42-3	Drops outline editing tables in calling the user's schema.
REFRESH_PRIVATE_OUTLINE Procedure on page 42-3	Refreshes the in-memory copy of the outline, synchronizing its data with the edits made to the outline hints.

CHANGE_JOIN_POS Procedure

This function changes the join position for the hint identified by outline name and hint number to the position specified by newpos.

Syntax

```
DBMS_OUTLN_EDIT.CHANGE_JOIN_POS (
   name     VARCHAR2
   hintno   NUMBER
   newpos   NUMBER);
```

Parameters

Table 42–2 CHANGE_JOIN_POS Procedure Parameters

Parameter	Description
name	Name of the private outline to be modified.
hintno	Hint number to be modified.
newpos	New join position for the target hint.

CREATE_EDIT_TABLES Procedure

This procedure creates outline editing tables in calling a user's schema.

Syntax

```
DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES;
```

DROP_EDIT_TABLES Procedure

This procedure drops outline editing tables in calling the user's schema.

Syntax

```
DBMS_OUTLN_EDIT.DROP_EDIT_TABLES;
```

REFRESH_PRIVATE_OUTLINE Procedure

This procedure refreshes the in-memory copy of the outline, synchronizing its data with the edits made to the outline hints.

Syntax

```
DBMS_OUTLN_EDIT.REFRESH_PRIVATE_OUTLINE (
   name  IN  VARCHAR2);
```

Parameters

Table 42–3 REFRESH_PRIVATE_OUTLINE Procedure Parameters

Parameter	Description
name	Name of the private outline to be refreshed.

43

DBMS_OUTPUT

The DBMS_OUTPUT package enables you to send messages from stored procedures, packages, and triggers.

The PUT and PUT_LINE procedures in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the GET_LINE procedure.

If you do not call GET_LINE, or if you do not display the messages on your screen in SQL*Plus or Enterprise Manager, then the buffered messages are ignored. The DBMS_OUTPUT package is especially useful for displaying PL/SQL debugging information.

Note: Messages sent using DBMS_OUTPUT are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

This chapter discusses the following topics:

Security, Errors, and Types for DBMS_OUTPUT

Security

At the end of this script, a public synonym (DBMS_OUTPUT) is created and EXECUTE permission on this package is granted to public.

Errors

DBMS_OUTPUT subprograms raise the application error ORA-20000, and the output procedures can return the following errors:

Table 43–1 DBMS_OUTPUT Errors

Error	Description
ORU-10027:	Buffer overflow
ORU-10028:	Line length overflow

Types

Type CHARARR is a table type.

Using DBMS_OUTPUT

A trigger might want to print out some debugging information. To do this, the trigger would do:

```
DBMS_OUTPUT.PUT_LINE('I got here:'||:new.col||' is the new value');
```

If you have enabled the DBMS_OUTPUT package, then this PUT_LINE would be buffered, and you could, after executing the statement (presumably some INSERT, DELETE, or UPDATE that caused the trigger to fire), get the line of information back. For example:

```
BEGIN
   DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;
```

It could then display the buffer on the screen. You repeat calls to GET_LINE until status comes back as nonzero. For better performance, you should use calls to GET_LINES which can return an array of lines.

Enterprise Manager and SQL*Plus implement a SET SERVEROUTPUT ON command to know whether to make calls to GET_LINE(S) after issuing INSERT, UPDATE, DELETE or anonymous PL/SQL calls (these are the only ones that can cause triggers or stored procedures to be executed).

Summary of DBMS_OUTPUT Subprograms

Table 43–2 DBMS_OUTPUT Package Subprograms

Subprogram	Description
ENABLE Procedure on page 43-3	Enables message output.
DISABLE Procedure on page 43-4	Disables message output.
PUT and PUT_LINE Procedures on page 43-4	PUT: Places a line in the buffer. PUT_LINE: Places partial line in buffer.
NEW_LINE Procedure on page 43-6	Terminates a line created with PUT.
GET_LINE and GET_LINES Procedures on page 43-6	Retrieves one line, or an array of lines, from buffer.

ENABLE Procedure

This procedure enables calls to PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. Calls to these procedures are ignored if the DBMS_OUTPUT package is not enabled.

> **Note:** It is not necessary to call this procedure when you use the SERVEROUTPUT option of Enterprise Manager or SQL*Plus.

If there are multiple calls to ENABLE, then buffer_size is the largest of the values specified. The maximum size is 1,000,000, and the minimum is 2,000.

Syntax

```
DBMS_OUTPUT.ENABLE (
   buffer_size IN INTEGER DEFAULT 20000);
```

Parameters

Table 43–3 ENABLE Procedure Parameters

Parameter	Description
buffer_size	Amount of information, in bytes, to buffer.

Pragmas

```
pragma restrict_references(enable,WNDS,RNDS);
```

Errors

Table 43–4 ENABLE Procedure Errors

Error	Description
ORA-20000:, ORU-10027:	Buffer overflow, limit of <buffer_limit> bytes.

DISABLE Procedure

This procedure disables calls to PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and purges the buffer of any remaining information.

As with ENABLE, you do not need to call this procedure if you are using the SERVEROUTPUT option of Enterprise Manager or SQL*Plus.

Syntax

```
DBMS_OUTPUT.DISABLE;
```

Pragmas

```
pragma restrict_references(disable,WNDS,RNDS);
```

PUT and PUT_LINE Procedures

You can either place an entire line of information into the buffer by calling PUT_LINE, or you can build a line of information piece by piece by making multiple calls to PUT. Both of these procedures are overloaded to accept items of type VARCHAR2, NUMBER, or DATE to place in the buffer.

All items are converted to VARCHAR2 as they are retrieved. If you pass an item of type NUMBER or DATE, then when that item is retrieved, it is formatted with TO_CHAR using the default format. If you want to use a different format, then you should pass in the item as VARCHAR2 and format it explicitly.

When you call PUT_LINE, the item that you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.

If your line exceeds the buffer limit, then you receive an error message.

Note: Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, Enterprise Manager or SQL*Plus do not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program. For example:

```
SQL> SET SERVER OUTPUT ON
SQL> BEGIN
     2 DBMS_OUTPUT.PUT_LINE ('hello');
     3 DBMS_LOCK.SLEEP (10);
     4 END;
```

Syntax

```
DBMS_OUTPUT.PUT      (item IN NUMBER);
DBMS_OUTPUT.PUT      (item IN VARCHAR2);
DBMS_OUTPUT.PUT      (item IN DATE);
DBMS_OUTPUT.PUT_LINE (item IN NUMBER);
DBMS_OUTPUT.PUT_LINE (item IN VARCHAR2);
DBMS_OUTPUT.PUT_LINE (item IN DATE);
DBMS_OUTPUT.NEW_LINE;
```

Parameters

Table 43–5 PUT and PUT_LINE Procedure Parameters

Parameter	Description
item	Item to buffer.

Errors

Table 43–6 PUT and PUT_LINE Procedure Errors

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 255 bytes per line.

NEW_LINE Procedure

This procedure puts an end-of-line marker. GET_LINE(S) returns "lines" as delimited by "newlines". Every call to PUT_LINE or NEW_LINE generates a line that is returned by GET_LINE(S).

Syntax

```
DBMS_OUTPUT.NEW_LINE;
```

Errors

Table 43–7 NEW_LINE Procedure Errors

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 255 bytes per line.

GET_LINE and GET_LINES Procedures

You can choose to retrieve from the buffer a single line or an array of lines. Call the GET_LINE procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the GET_LINES procedure to retrieve an array of lines from the buffer.

You can choose to automatically display this information if you are using Enterprise Manager or SQL*Plus by using the special SET SERVEROUTPUT ON command.

After calling GET_LINE or GET_LINES, any lines not retrieved before the next call to PUT, PUT_LINE, or NEW_LINE are discarded to avoid confusing them with the next message.

Syntax

```
DBMS_OUTPUT.GET_LINE (
   line    OUT VARCHAR2,
   status  OUT INTEGER);
```

Parameters

Table 43–8 GET_LINE Procedure Parameters

Parameter	Description
line	Returns a single line of buffered information, excluding a final newline character: The maximum length is 255 bytes.
status	If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.

Syntax

```
DBMS_OUTPUT.GET_LINES (
   lines       OUT  CHARARR,
   numlines    IN OUT  INTEGER);
```

CHARARR is a table of VARCHAR2(255).

Parameters

Table 43–9 GET_LINES Procedure Parameters

Parameter	Description
lines	Returns an array of lines of buffered information. The maximum length of each line in the array is 255 bytes.
numlines	Number of lines you want to retrieve from the buffer. After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.

Example 1: Debugging Stored Procedures and Triggers

The DBMS_OUTPUT package is commonly used to debug stored procedures and triggers. This package can also be used to enable you to retrieve information about

an object and format this output, as shown in "Example 2: Retrieving Information About an Object" on page 43-9.

This function queries the employee table and returns the total salary for a specified department. The function includes several calls to the PUT_LINE procedure:

```
CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
   CURSOR emp_cursor IS
      SELECT sal, comm FROM emp WHERE deptno = dnum;
   total_wages    NUMBER(11, 2) := 0;
   counter        NUMBER(10) := 1;
BEGIN

   FOR emp_record IN emp_cursor LOOP
      emp_record.comm := NVL(emp_record.comm, 0);
      total_wages := total_wages + emp_record.sal
         + emp_record.comm;
      DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
         '; Wages = '|| TO_CHAR(total_wages));    /* Debug line */
      counter := counter + 1; /* Increment debug counter */
   END LOOP;
   /* Debug line */
   DBMS_OUTPUT.PUT_LINE('Total wages = ' ||
     TO_CHAR(total_wages));
   RETURN total_wages;

END dept_salary;
```

Assume the EMP table contains the following rows:

```
EMPNO          SAL     COMM     DEPT
-----        ------- -------- -------
1002           1500      500       20
1203           1000                30
1289           1000                10
1347           1000      250       20
```

Assume the user executes the following statements in the Enterprise Manager SQL Worksheet input pane:

```
SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);
```

The user would then see the following information displayed in the output pane:

```
Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Total wages = 3250


PL/SQL procedure successfully executed.
```

Example 2: Retrieving Information About an Object

In this example, the user has used the EXPLAIN PLAN command to retrieve information about the execution plan for a statement and has stored it in PLAN_TABLE. The user has also assigned a statement ID to this statement. The example EXPLAIN_OUT procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```
 /*****************************************************************/
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* to procedure, to uniquely identify statement.                 */
/*****************************************************************/
CREATE OR REPLACE PROCEDURE explain_out
   (statement_id IN VARCHAR2) AS

   -- Retrieve information from PLAN_TABLE into cursor EXPLAIN_ROWS.

   CURSOR explain_rows IS
      SELECT level, id, position, operation, options,
         object_name
      FROM plan_table
      WHERE statement_id = explain_out.statement_id
      CONNECT BY PRIOR id = parent_id
         AND statement_id = explain_out.statement_id
      START WITH id = 0
       ORDER BY id;

BEGIN


   -- Loop through information retrieved from PLAN_TABLE:

   FOR line IN explain_rows LOOP

      -- At start of output, include heading with estimated cost.

      IF line.id = 0 THEN
         DBMS_OUTPUT.PUT_LINE ('Plan for statement '
```

```
            || statement_id
            || ', estimated cost = ' || line.position);
      END IF;

      -- Output formatted information. LEVEL determines indention level.

      DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) ||
         line.operation || ' ' || line.options || ' ' ||
         line.object_name);
   END LOOP;

END;
```

See Also: Chapter 95, "UTL_FILE"

44

DBMS_PCLXUTIL

The DBMS_PCLXUTIL package provides intra-partition parallelism for creating partition-wise local indexes.

> **See Also:** There are several rules concerning partitions and indexes. For more information, see *Oracle9i Database Concepts* and *Oracle9i Database Administrator's Guide.*

DBMS_PCLXUTIL circumvents the limitation that, for local index creation, the degree of parallelism is restricted to the number of partitions as only one slave process for each partition is used.

DBMS_PCLXUTIL uses the DBMS_JOB package to provide a greater degree of parallelism for creating a local index for a partitioned table. This is achieved by asynchronous inter-partition parallelism using the background processes (with DBMS_JOB), in combination with intra-partition parallelism using the parallel query slave processes.

DBMS_PCLXUTIL works with both range and range-hash composite partitioning.

> **Note:** For range partitioning, the minimum compatibility mode is 8.0; for range-hash composite partitioning, the minimum compatibility mode is 8*i*.

This chapter discusses the following topics:

Using DBMS_PCLXUTIL

The DBMS_PCLXUTIL package can be used during the following DBA tasks:

1. Local index creation

   The procedure BUILD_PART_INDEX assumes that the dictionary information for the local index already exists. This can be done by issuing the create index SQL command with the UNUSABLE option.

   ```
   CREATE INDEX <idx_name> on <tab_name>(...) local(...) unusable;
   ```

   This causes the dictionary entries to be created without "building" the index itself, the time consuming part of creating an index. Now, invoking the procedure BUILD_PART_INDEX causes a concurrent build of local indexes with the specified degree of parallelism.

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
   ```

   For composite partitions, the procedure automatically builds local indexes for all subpartitions of the composite table.

2. Local index maintenance

   By marking desired partitions usable or unusable, the BUILD_PART_INDEX procedure also enables selective rebuilding of local indexes. The force_opt parameter provides a way to override this and build local indexes for all partitions.

   ```
   ALTER INDEX <idx_name> local(...) unusable;
   ```

   Rebuild only the desired (sub)partitions (that are marked unusable):

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
   ```

   Rebuild all (sub)partitions using force_opt = TRUE:

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,TRUE);
   ```

   A progress report is produced, and the output appears on screen when the program is ended (because the DBMS_OUTPUT package writes messages to a buffer first, and flushes the buffer to the screen only upon termination of the program).

Limitations

Because DBMS_PCLXUTIL uses the DBMS_JOB package, you must be aware of the following limitations pertaining to DBMS_JOB:

- You must decide appropriate values for the job_queue_processes initalization parameter. Clearly, if the job processes are not started before calling BUILD_PART_INDEX(), then the package will not function properly. The background processes are specified by the following init.ora parameters:

  ```
  job_queue_processes=n   #the number of background processes = n
  ```

- There is an upper limit to the number of simultaneous jobs in the queue, dictated by the upper limit on the number of background processes marked SNP[0..9] and SNP[A..Z], which is 36.

  See Also: *Oracle9i Database Administrator's Guide*

- Failure conditions are reported only in the trace files (a DBMS_JOB limitation), making it impossible to give interactive feedback to the user. This package prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the snp*.trc trace files.

Summary of DBMS_PCLUTTL Subprograms

DBMS_PCLXUTIL contains just one procedure: BUILD_PART_INDEX.

BUILD_PART_INDEX Procedure

Syntax

```
DBMS_PCLXUTIL.build_part_index (
   procs_per_job   IN NUMBER   DEFAULT 1,
   tab_name        IN VARCHAR2 DEFAULT NULL,
   idx_name        IN VARCHAR2 DEFAULT NULL,
   force_opt       IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 44–1 BUILD_PART_INDEX Procedure Parameters

Parameter	Description
procs_per_job	Number of parallel query slaves to be utilized for each local index build (1 <= procs_per_job <= max_slaves).
tab_name	Name of the partitioned table (an exception is raised if the table does not exist or not partitioned).
idx_name	Name given to the local index (an exception is raised if a local index is not created on the table tab_name).
force_opt	If TRUE, then force rebuild of all partitioned indexes; otherwise, rebuild only the partitions marked 'UNUSABLE'.

Example

Suppose a table PROJECT is created with two partitions PROJ001 and PROJ002, along with a local index IDX.

A call to the procedure BUILD_PART_INDEX(2,4,'PROJECT','IDX',TRUE) produces the following output:

```
SQLPLUS> EXECUTE dbms_pclxutil.build_part_index(2,4,'PROJECT','IDX',TRUE);
Statement processed.
INFO: Job #21 created for partition PROJ002 with 4 slaves
INFO: Job #22 created for partition PROJ001 with 4 slaves
```

45

DBMS_PIPE

The DBMS_PIPE package lets two or more sessions in the same instance communicate. Oracle pipes are similar in concept to the pipes used in UNIX, but Oracle pipes are not implemented using the operating system pipe mechanisms.

Information sent through Oracle pipes is buffered in the system global area (SGA). All information in pipes is lost when the instance is shut down.

Depending upon your security requirements, you may choose to use either a *public* or a *private* pipe.

> **Caution:** Pipes are independent of transactions. Be careful using pipes when transaction control can be affected.

This chapter discusses the following topics:

- Public Pipes, Private Pipes, and Pipe Uses
- Security, Constants, and Errors
- Summary of DBMS_PIPE Subprograms

Public Pipes, Private Pipes, and Pipe Uses

Public Pipes

You may create a public pipe either implicitly or explicitly. For *implicit* public pipes, the pipe is automatically created when it is referenced for the first time, and it disappears when it no longer contains data. Because the pipe descriptor is stored in the SGA, there is some space usage overhead until the empty pipe is aged out of the cache.

You create an *explicit* public pipe by calling the CREATE_PIPE function with the `private` flag set to FALSE. You must deallocate explicitly-created pipes by calling the REMOVE_PIPE function.

The domain of a public pipe is the schema in which it was created, either explicitly or implicitly.

Writing and Reading Pipes

Each public pipe works asynchronously. Any number of schema users can write to a public pipe, as long as they have EXECUTE permission on the DBMS_PIPE package, and they know the name of the public pipe. However, once buffered information is read by one user, it is emptied from the buffer, and is not available for other readers of the same pipe.

The sending session builds a message using one or more calls to the PACK_MESSAGE procedure. This procedure adds the message to the session's local message buffer. The information in this buffer is sent by calling the SEND_MESSAGE function, designating the pipe name to be used to send the message. When SEND_MESSAGE is called, all messages that have been stacked in the local buffer are sent.

A process that wants to receive a message calls the RECEIVE_MESSAGE function, designating the pipe name from which to receive the message. The process then calls the UNPACK_MESSAGE procedure to access each of the items in the message.

Private Pipes

You explicitly create a private pipe by calling the CREATE_PIPE function. Once created, the private pipe persists in shared memory until you explicitly deallocate it by calling the REMOVE_PIPE function. A private pipe is also deallocated when the database instance is shut down.

You cannot create a private pipe if an implicit pipe exists in memory and has the same name as the private pipe you are trying to create. In this case, CREATE_PIPE returns an error.

Access to a private pipe is restricted to:

- Sessions running under the same userid as the creator of the pipe
- Stored subprograms executing in the same userid privilege domain as the pipe creator
- Users connected as SYSDBA

An attempt by any other user to send or receive messages on the pipe, or to remove the pipe, results in an immediate error. Any attempt by another user to create a pipe with the same name also causes an error.

As with public pipes, you must first build your message using calls to PACK_MESSAGE before calling SEND_MESSAGE. Similarly, you must call RECEIVE_MESSAGE to retrieve the message before accessing the items in the message by calling UNPACK_MESSAGE.

Pipe Uses

The pipe functionality has several potential applications:

- External service interface: You can communicate with user-written services that are external to the RDBMS. This can be done effectively in a shared server process, so that several instances of the service are executing simultaneously. Additionally, the services are available asynchronously. The requestor of the service does not need to block a waiting reply. The requestor can check (with or without timeout) at a later time. The service can be written in any of the 3GL languages that Oracle supports.
- Independent transactions: The pipe can communicate to a separate session which can perform an operation in an independent transaction (such as logging an attempted security violation detected by a trigger).
- Alerters (non-transactional): You can post another process without requiring the waiting process to poll. If an "after-row" or "after-statement" trigger were to alert an application, then the application would treat this alert as an indication that the data probably changed. The application would then read the data to get the current value. Because this is an "after" trigger, the application would want to do a "select for update" to make sure it read the correct data.
- Debugging: Triggers and stored procedures can send debugging information to a pipe. Another session can keep reading out of the pipe and display it on the screen or write it to a file.

- Concentrator: This is useful for multiplexing large numbers of users over a fewer number of network connections, or improving performance by concentrating several user-transactions into one DBMS transaction.

Security, Constants, and Errors

Security

Security can be achieved by use of `GRANT EXECUTE` on the `DBMS_PIPE` package by creating a pipe using the `private` parameter in the `CREATE_PIPE` function and by writing cover packages that only expose particular features or pipenames to particular users or roles.

Constants

```
maxwait   constant integer := 86400000; /* 1000 days */
```

This is the maximum time to wait attempting to send or receive a message.

Errors

`DBMS_PIPE` package subprograms can return the following errors:

Table 45–1 DBMS_PIPE Errors

Error	Description
`ORA-23321:`	Pipename may not be null. This can be returned by the `CREATE_PIPE` function, or any subprogram that takes a pipe name as a parameter.
`ORA-23322:`	Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.

Summary of DBMS_PIPE Subprograms

Table 45–2 DBMS_PIPE Package Subprograms

Subprogram	Description
CREATE_PIPE Function on page 45-5	Creates a pipe (necessary for private pipes).
PACK_MESSAGE Procedure on page 45-7	Builds message in local buffer.

Table 45–2 DBMS_PIPE Package Subprograms (Cont.)

Subprogram	Description
SEND_MESSAGE Function on page 45-8	Sends message on named pipe: This implicitly creates a public pipe if the named pipe does not exist.
RECEIVE_MESSAGE Function on page 45-10	Copies message from named pipe into local buffer.
NEXT_ITEM_TYPE Function on page 45-12	Returns datatype of next item in buffer.
UNPACK_MESSAGE Procedure on page 45-13	Accesses next item in buffer.
REMOVE_PIPE Function on page 45-14	Removes the named pipe.
PURGE Procedure on page 45-15	Purges contents of named pipe.
RESET_BUFFER Procedure on page 45-16	Purges contents of local buffer.
UNIQUE_SESSION_NAME Function on page 45-16	Returns unique session name.

CREATE_PIPE Function

This function explicitly creates a public or private pipe. If the `private` flag is `TRUE`, then the pipe creator is assigned as the owner of the private pipe.

Explicitly-created pipes can only be removed by calling `REMOVE_PIPE`, or by shutting down the instance.

Syntax

```
DBMS_PIPE.CREATE_PIPE (
   pipename     IN VARCHAR2,
   maxpipesize  IN INTEGER DEFAULT 8192,
   private      IN BOOLEAN DEFAULT TRUE)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(create_pipe,WNDS,RNDS);
```

Parameters

Table 45–3 CREATE_PIPE Function Parameters

Parameter	Description
pipename	Name of the pipe you are creating. You must use this name when you call SEND_MESSAGE and RECEIVE_MESSAGE. This name must be unique across the instance. Caution: Do not use pipe names beginning with ORA$. These are reserved for use by procedures provided by Oracle Corporation. Pipename should not be longer than 128 bytes, and is case_insensitive. At this time, the name cannot contain NLS characters.
maxpipesize	The maximum size allowed for the pipe, in bytes. The total size of all of the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default maxpipesize is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value use the existing, larger value.
private	Uses the default, TRUE, to create a private pipe. Public pipes can be implicitly created when you call SEND_MESSAGE.

Returns

Table 45–4 CREATE_PIPE Function Returns

Return	Description
0	Successful. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBA/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.

Table 45–4 CREATE_PIPE Function Returns

Return	Description
ORA-23322	Failure due to naming conflict. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 45–5 CREATE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Pipe with the same name already exists, and you are not allowed to use it.

PACK_MESSAGE Procedure

This procedure builds your message in the local message buffer.

To send a message, first make one or more calls to PACK_MESSAGE. Then, call SEND_MESSAGE to send the message in the local buffer on the named pipe.

The PACK_MESSAGE procedure is overloaded to accept items of type VARCHAR2, NUMBER, or DATE. In addition to the data bytes, each item in the buffer requires one byte to indicate its type, and two bytes to store its length. One additional byte is needed to terminate the message.The overhead for all types other than VARCHAR is 4 bytes.

In Oracle8, the char-set-id (2 bytes) and the char-set-form (1 byte) are stored with each data item. Therefore, the overhead when using Oracle8 is 7 bytes.

When you call SEND_MESSAGE to send this message, you must indicate the name of the pipe on which you want to send the message. If this pipe already exists, then you must have sufficient privileges to access this pipe. If the pipe does not already exist, then it is created automatically.

Syntax

```
DBMS_PIPE.PACK_MESSAGE       (item  IN  VARCHAR2);
DBMS_PIPE.PACK_MESSAGE       (item  IN  NCHAR);
DBMS_PIPE.PACK_MESSAGE       (item  IN  NUMBER);
```

```
DBMS_PIPE.PACK_MESSAGE       (item  IN  DATE);
DBMS_PIPE.PACK_MESSAGE_RAW   (item  IN  RAW);
DBMS_PIPE.PACK_MESSAGE_ROWID (item  IN  ROWID);
```

Note: The PACK_MESSAGE procedure is overloaded to accept items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to pack RAW and ROWID items.

Pragmas

```
pragma restrict_references(pack_message,WNDS,RNDS);
pragma restrict_references(pack_message_raw,WNDS,RNDS);
pragma restrict_references(pack_message_rowid,WNDS,RNDS);
```

Parameters

Table 45–6 PACK_MESSAGE Procedure Parameters

Parameter	Description
item	Item to pack into the local message buffer.

Exceptions

ORA-06558 is raised if the message buffer overflows (currently 4096 bytes). Each item in the buffer takes one byte for the type, two bytes for the length, plus the actual data. There is also one byte needed to terminate the message.

SEND_MESSAGE Function

This function sends a message on the named pipe.

The message is contained in the local message buffer, which was filled with calls to PACK_MESSAGE. A pipe could be explicitly using CREATE_PIPE; otherwise, it is created implicitly.

Syntax

```
DBMS_PIPE.SEND_MESSAGE (
    pipename     IN VARCHAR2,
    timeout      IN INTEGER DEFAULT MAXWAIT,
    maxpipesize  IN INTEGER DEFAULT 8192)
```

```
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(send_message,WNDS,RNDS);
```

Parameters

Table 45–7 SEND_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to place the message. If you are using an explicit pipe, then this is the name that you specified when you called CREATE_PIPE. Caution: Do not use pipe names beginning with 'ORA$'. These names are reserved for use by procedures provided by Oracle Corporation. Pipename should not be longer than 128 bytes, and is case-insensitive. At this time, the name cannot contain NLS characters.
timeout	Time to wait while attempting to place a message on a pipe, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days).
maxpipesize	Maximum size allowed for the pipe, in bytes. The total size of all the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value simply use the existing, larger value. Specifying maxpipesize as part of the SEND_MESSAGE procedure eliminates the need for a separate call to open the pipe. If you created the pipe explicitly, then you can use the optional maxpipesize parameter to override the creation pipe size specifications.

Returns

Table 45–8 SEND_MESSAGE Function Returns

Return	Description
0	Success. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBS/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.
1	Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used. If the pipe was implicitly-created and is empty, then it is removed.
3	An interrupt occurred. If the pipe was implicitly created and is empty, then it is removed.
ORA-23322	Insufficient privileges. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 45–9 SEND_MESSAGE Function Exception

Exception	Description
Null pipe name	Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.

RECEIVE_MESSAGE Function

This function copies the message into the local message buffer.

To receive a message from a pipe, first call RECEIVE_MESSAGE. When you receive a message, it is removed from the pipe; hence, a message can only be received once. For implicitly-created pipes, the pipe is removed after the last record is removed from the pipe.

If the pipe that you specify when you call RECEIVE_MESSAGE does not already exist, then Oracle implicitly creates the pipe and waits to receive the message. If the message does not arrive within a designated timeout interval, then the call returns and the pipe is removed.

After receiving the message, you must make one or more calls to UNPACK_MESSAGE to access the individual items in the message. The UNPACK_MESSAGE procedure is overloaded to unpack items of type DATE, NUMBER, VARCHAR2, and there are two additional procedures to unpack RAW and ROWID items. If you do not know the type of data that you are attempting to unpack, then call NEXT_ITEM_TYPE to determine the type of the next item in the buffer.

Syntax

```
DBMS_PIPE.RECEIVE_MESSAGE (
   pipename     IN VARCHAR2,
   timeout      IN INTEGER      DEFAULT maxwait)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(receive_message,WNDS,RNDS);
```

Parameters

Table 45–10 RECEIVE_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to receive a message. Names beginning with ORA$ are reserved for use by Oracle
timeout	Time to wait for a message, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days). A timeout of 0 allows you to read without blocking.

Returns

Table 45–11 RECEIVE_MESSAGE Function Returns

Return	Description
0	Success

Table 45–11 RECEIVE_MESSAGE Function Returns

Return	Description
1	Timed out. If the pipe was implicitly-created and is empty, then it is removed.
2	Record in the pipe is too large for the buffer. (This should not happen.)
3	An interrupt occurred.
ORA-23322	User has insufficient privileges to read from the pipe.

Exceptions

Table 45–12 RECEIVE_MESSAGE Function Exceptions

Exception	Description
Null pipe name	Permission error. Insufficient privilege to remove the record from the pipe. The pipe is owned by someone else.

NEXT_ITEM_TYPE Function

This function determines the datatype of the next item in the local message buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call NEXT_ITEM_TYPE.

Syntax

```
DBMS_PIPE.NEXT_ITEM_TYPE
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(next_item_type,WNDS,RNDS);
```

Returns

Table 45–13 NEXT_ITEM_TYPE Function Returns

Return	Description
0	No more items

Table 45–13 NEXT_ITEM_TYPE Function Returns

Return	Description
6	NUMBER
9	VARCHAR2
11	ROWID
12	DATE
23	RAW

UNPACK_MESSAGE Procedure

This procedure retrieves items from the buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call UNPACK_MESSAGE.

Syntax

```
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT VARCHAR2);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT NCHAR);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT NUMBER);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT DATE);
DBMS_PIPE.UNPACK_MESSAGE_RAW   (item  OUT RAW);
DBMS_PIPE.UNPACK_MESSAGE_ROWID (item  OUT ROWID);
```

Note: The UNPACK_MESSAGE procedure is overloaded to return items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to unpack RAW and ROWID items.

Pragmas

```
pragma restrict_references(unpack_message,WNDS,RNDS);
pragma restrict_references(unpack_message_raw,WNDS,RNDS);
pragma restrict_references(unpack_message_rowid,WNDS,RNDS);
```

Parameters

Table 45–14 UNPACK_MESSAGE Procedure Parameters

Parameter	Description
item	Argument to receive the next unpacked item from the local message buffer.

Exceptions

ORA-06556 or 06559 are generated if the buffer contains no more items, or if the item is not of the same type as that requested.

REMOVE_PIPE Function

This function removes explicitly-created pipes.

Pipes created implicitly by SEND_MESSAGE are automatically removed when empty. However, pipes created explicitly by CREATE_PIPE are removed only by calling REMOVE_PIPE, or by shutting down the instance. All unconsumed records in the pipe are removed before the pipe is deleted.

This is similar to calling PURGE on an implicitly-created pipe.

Syntax

```
DBMS_PIPE.REMOVE_PIPE (
   pipename  IN  VARCHAR2)
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(remove_pipe,WNDS,RNDS);
```

Parameters

Table 45–15 REMOVE_PIPE Function Parameters

Parameter	Description
pipename	Name of pipe that you want to remove.

Returns

Table 45–16 REMOVE_PIPE Function Returns

Return	Description
0	Success If the pipe does not exist, or if the pipe already exists and the user attempting to remove it is authorized to do so, then Oracle returns 0, indicating success, and any data remaining in the pipe is removed.
ORA-23322	Insufficient privileges. If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA-23322, indicating insufficient privileges.

Exceptions

Table 45–17 REMOVE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.

PURGE Procedure

This procedure empties the contents of the named pipe.

An empty implicitly-created pipe is aged out of the shared global area according to the least-recently-used algorithm. Thus, calling PURGE lets you free the memory associated with an implicitly-created pipe.

Because PURGE calls RECEIVE_MESSAGE, the local buffer might be overwritten with messages as they are purged from the pipe. Also, you can receive an ORA-23322 (insufficient privileges) error if you attempt to purge a pipe with which you have insufficient access rights.

Syntax

```
DBMS_PIPE.PURGE (
   pipename  IN  VARCHAR2);
```

Pragmas

```
pragma restrict_references(purge,WNDS,RNDS);
```

Parameters

Table 45–18 Purge Procedure Parameters

Parameter	Description
pipename	Name of pipe from which to remove all messages. The local buffer may be overwritten with messages as they are discarded. Pipename should not be longer than 128 bytes, and is case-insensitive.

Exceptions

Permission error if pipe belongs to another user.

RESET_BUFFER Procedure

This procedure resets the PACK_MESSAGE and UNPACK_MESSAGE positioning indicators to 0.

Because all pipes share a single buffer, you may find it useful to reset the buffer before using a new pipe. This ensures that the first time you attempt to send a message to your pipe, you do not inadvertently send an expired message remaining in the buffer.

Syntax

```
DBMS_PIPE.RESET_BUFFER;
```

Pragmas

```
pragma restrict_references(reset_buffer,WNDS,RNDS);
```

UNIQUE_SESSION_NAME Function

This function receives a name that is unique among all of the sessions that are currently connected to a database.

Multiple calls to this function from the same session always return the same value. You might find it useful to use this function to supply the PIPENAME parameter for your SEND_MESSAGE and RECEIVE_MESSAGE calls.

Syntax

```
DBMS_PIPE.UNIQUE_SESSION_NAME
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(unique_session_name,WNDS,RNDS,WNPS);
```

Returns

This function returns a unique name. The returned name can be up to 30 bytes.

Example 1: Debugging

This example shows the procedure that a PL/SQL program can call to place debugging information in a pipe.

```
CREATE OR REPLACE PROCEDURE debug (msg VARCHAR2) AS
    status  NUMBER;
BEGIN
  DBMS_PIPE.PACK_MESSAGE(LENGTH(msg));
  DBMS_PIPE.PACK_MESSAGE(msg);
  status := DBMS_PIPE.SEND_MESSAGE('plsql_debug');
  IF status != 0 THEN
    raise_application_error(-20099, 'Debug error');
  END IF;
END debug;
```

The following Pro*C code receives messages from the PLSQL_DEBUG pipe in "Example 1: Debugging" and displays the messages. If the Pro*C session is run in a separate window, then it can be used to display any messages that are sent to the debug procedure from a PL/SQL program executing in a separate session.

```
#include <stdio.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR username[20];
   int     status;
   int     msg_length;
   char    retval[2000];
```

```
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void sql_error();

main()
{

-- Prepare username:
   strcpy(username.arr, "SCOTT/TIGER");
   username.len = strlen(username.arr);

   EXEC SQL WHENEVER SQLERROR DO sql_error();
   EXEC SQL CONNECT :username;

   printf("connected\n");

-- Start an endless loop to look for and print messages on the pipe:
   FOR (;;)
   {
      EXEC SQL EXECUTE
         DECLARE
            len INTEGER;
            typ INTEGER;
            sta INTEGER;
            chr VARCHAR2(2000);
         BEGIN
            chr := '';
            sta := dbms_pipe.receive_message('plsql_debug');
            IF sta = 0 THEN
               DBMS_PIPE.UNPACK_MESSAGE(len);
               DBMS_PIPE.UNPACK_MESSAGE(chr);
            END IF;
            :status := sta;
            :retval := chr;
            IF len IS NOT NULL THEN
               :msg_length := len;
            ELSE
               :msg_length := 2000;
            END IF;
         END;
      END-EXEC;
      IF (status == 0)
         printf("\n%.*s\n", msg_length, retval);
```

```
      ELSE
         printf("abnormal status, value is %d\n", status);
   }
}

void sql_error()
{
   char msg[1024];
   int rlen, len;
   len = sizeof(msg);
   sqlglm(msg, &len, &rlen);
   printf("ORACLE ERROR\n");
   printf("%.*s\n", rlen, msg);
   exit(1);
}
```

Example 2: Execute System Commands

This example shows PL/SQL and Pro*C code let a PL/SQL stored procedure (or anonymous block) call PL/SQL procedures to send commands over a pipe to a Pro*C program that is listening for them.

The Pro*C program sleeps and waits for a message to arrive on the named pipe. When a message arrives, the C program processes it, carrying out the required action, such as executing a UNIX command through the *system*() call or executing a SQL command using embedded SQL.

`DAEMON.SQL` is the source code for the PL/SQL package. This package contains procedures that use the `DBMS_PIPE` package to send and receive message to and from the Pro*C daemon. Note that full handshaking is used. The daemon always sends a message back to the package (except in the case of the `STOP` command). This is valuable, because it allows the PL/SQL procedures to be sure that the Pro*C daemon is running.

You can call the DAEMON packaged procedures from an anonymous PL/SQL block using SQL*Plus or Enterprise Manager. For example:

```
SQLPLUS> variable rv number
SQLPLUS> execute :rv := DAEMON.EXECUTE_SYSTEM('ls -la');
```

On a UNIX system, this causes the Pro*C daemon to execute the command *system*("*ls -la*").

Remember that the daemon needs to be running first. You might want to run it in the background, or in another window beside the SQL*Plus or Enterprise Manager session from which you call it.

The DAEMON.SQL also uses the DBMS_OUTPUT package to display the results. For this example to work, you must have execute privileges on this package.

DAEMON.SQL Example. This is the code for the PL/SQL DAEMON package:

```
CREATE OR REPLACE PACKAGE daemon AS
  FUNCTION execute_sql(command VARCHAR2,
                       timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  PROCEDURE stop(timeout NUMBER DEFAULT 10);
END daemon;
/
CREATE OR REPLACE PACKAGE BODY daemon AS

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
  RETURN NUMBER IS

    status       NUMBER;
    result       VARCHAR2(20);
    command_code NUMBER;
    pipe_name    VARCHAR2(30);
  BEGIN
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

    DBMS_PIPE.PACK_MESSAGE('SYSTEM');
    DBMS_PIPE.PACK_MESSAGE(pipe_name);
    DBMS_PIPE.PACK_MESSAGE(command);
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20010,
        'Execute_system: Error while sending.  Status = ' ||
         status);
    END IF;

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);
    IF status <> 0 THEN
```

```
    RAISE_APPLICATION_ERROR(-20011,
      'Execute_system: Error while receiving.
       Status = ' || status);
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(result);
  IF result <> 'done' THEN
    RAISE_APPLICATION_ERROR(-20012,
      'Execute_system: Done not received.');
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(command_code);
  DBMS_OUTPUT.PUT_LINE('System command executed.  result = ' ||
                       command_code);
  RETURN command_code;
END execute_system;

FUNCTION execute_sql(command VARCHAR2,
                     timeout NUMBER DEFAULT 10)
RETURN NUMBER IS

  status       NUMBER;
  result       VARCHAR2(20);
  command_code NUMBER;
  pipe_name    VARCHAR2(30);

BEGIN
  pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

  DBMS_PIPE.PACK_MESSAGE('SQL');
  DBMS_PIPE.PACK_MESSAGE(pipe_name);
  DBMS_PIPE.PACK_MESSAGE(command);
  status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20020,
      'Execute_sql: Error while sending.  Status = ' || status);
  END IF;

  status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);

  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20021,
      'execute_sql: Error while receiving.
       Status = ' || status);
  END IF;
```

```
    DBMS_PIPE.UNPACK_MESSAGE(result);
    IF result <> 'done' THEN
      RAISE_APPLICATION_ERROR(-20022,
        'execute_sql: done not received.');
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(command_code);
    DBMS_OUTPUT.PUT_LINE
        ('SQL command executed.  sqlcode = ' || command_code);
    RETURN command_code;
  END execute_sql;

  PROCEDURE stop(timeout NUMBER DEFAULT 10) IS
    status NUMBER;
  BEGIN
    DBMS_PIPE.PACK_MESSAGE('STOP');
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20030,
        'stop: error while sending.  status = ' || status);
    END IF;
  END stop;
END daemon;
```

daemon.pc Example. This is the code for the Pro*C daemon. You must precompile this using the Pro*C Precompiler, Version 1.5.x or later. You must also specify the USERID and SQLCHECK options, as the example contains embedded PL/SQL code.

Note: To use a VARCHAR output host variable in a PL/SQL block, you must initialize the length component before entering the block.

```
proc iname=daemon userid=scott/tiger sqlcheck=semantics
```

Then C-compile and link in the normal way.

```
#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;
```

```
EXEC SQL BEGIN DECLARE SECTION;
  char *uid = "scott/tiger";
  int status;
  VARCHAR command[20];
  VARCHAR value[2000];
  VARCHAR return_name[30];
EXEC SQL END DECLARE SECTION;

void
connect_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while connecting:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon quitting.\n");
  exit(1);
}

void
sql_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while executing:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon continuing.\n");
}
main()
{
command.len = 20; /*initialize length components*/
value.len = 2000;
return_name.len  = 30;
  EXEC SQL WHENEVER SQLERROR DO connect_error();
  EXEC SQL CONNECT :uid;
  printf("Daemon connected.\n");
```

```
EXEC SQL WHENEVER SQLERROR DO sql_error();
printf("Daemon waiting...\n");
while (1) {
  EXEC SQL EXECUTE
    BEGIN
      :status := DBMS_PIPE.RECEIVE_MESSAGE('daemon');
      IF :status = 0 THEN
        DBMS_PIPE.UNPACK_MESSAGE(:command);
      END IF;
    END;
  END-EXEC;
  IF (status == 0)
  {
    command.arr[command.len] = '\0';
    IF (!strcmp((char *) command.arr, "STOP"))
    {
      printf("Daemon exiting.\n");
      break;
    }

    ELSE IF (!strcmp((char *) command.arr, "SYSTEM"))
    {
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.UNPACK_MESSAGE(:return_name);
          DBMS_PIPE.UNPACK_MESSAGE(:value);
        END;
      END-EXEC;
      value.arr[value.len] = '\0';
      printf("Will execute system command '%s'\n", value.arr);

      status = system(value.arr);
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.PACK_MESSAGE('done');
          DBMS_PIPE.PACK_MESSAGE(:status);
          :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
        END;
      END-EXEC;

      IF (status)
      {
        printf
         ("Daemon error while responding to system command.");
        printf("  status: %d\n", status);
```

```
      }
    }
    ELSE IF (!strcmp((char *) command.arr, "SQL")) {
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.UNPACK_MESSAGE(:return_name);
          DBMS_PIPE.UNPACK_MESSAGE(:value);
        END;
      END-EXEC;
      value.arr[value.len] = '\0';
      printf("Will execute sql command '%s'\n", value.arr);

      EXEC SQL WHENEVER SQLERROR CONTINUE;
      EXEC SQL EXECUTE IMMEDIATE :value;
      status = sqlca.sqlcode;

      EXEC SQL WHENEVER SQLERROR DO sql_error();
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.PACK_MESSAGE('done');
          DBMS_PIPE.PACK_MESSAGE(:status);
          :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
        END;
      END-EXEC;

      IF (status)
      {
        printf("Daemon error while responding to sql command.");
        printf("  status: %d\n", status);
      }
    }
    ELSE
    {
      printf
        ("Daemon error: invalid command '%s' received.\n",
          command.arr);
    }
  }
  ELSE
  {
    printf("Daemon error while waiting for signal.");
    printf("  status = %d\n", status);
  }
}
EXEC SQL COMMIT WORK RELEASE;
```

```
exit(0);
```

Example 3: External Service Interface

Put the user-written 3GL code into an OCI or Precompiler program. The program connects to the database and executes PL/SQL code to read its request from the pipe, computes the result, and then executes PL/SQL code to send the result on a pipe back to the requestor.

Below is an example of a stock service request. The recommended sequence for the arguments to pass on the pipe for all service requests is:

```
protocol_version      VARCHAR2          - '1', 10 bytes or less
returnpipe            VARCHAR2          - 30 bytes or less
service               VARCHAR2          - 30 bytes or less
arg1                  VARCHAR2/NUMBER/DATE
   ...
argn                  VARCHAR2/NUMBER/DATE
```

The recommended format for returning the result is:

```
success               VARCHAR2          - 'SUCCESS' if OK,
                                          otherwise error message
arg1                  VARCHAR2/NUMBER/DATE
   ...
argn                  VARCHAR2/NUMBER/DATE
```

The "stock price request server" would do, using OCI or PRO* (in pseudo-code):

```
<loop forever>
  BEGIN dbms_stock_server.get_request(:stocksymbol); END;
  <figure out price based on stocksymbol (probably from some radio
        signal), set error if can't find such a stock>
  BEGIN dbms_stock_server.return_price(:error, :price); END;
```

A client would do:

```
BEGIN :price := stock_request('YOURCOMPANY'); end;
```

The stored procedure, `dbms_stock_server`, which is called by the preceding "stock price request server" is:

```
CREATE OR REPLACE PACKAGE dbms_stock_server IS
  PROCEDURE get_request(symbol OUT VARCHAR2);
  PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;
```

```
CREATE OR REPLACE PACKAGE BODY dbms_stock_server IS
  returnpipe    VARCHAR2(30);

  PROCEDURE returnerror(reason VARCHAR2) IS
    s INTEGER;
  BEGIN
    dbms_pipe.pack_message(reason);
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        ' sending on pipe');
    END IF;
  END;

  PROCEDURE get_request(symbol OUT VARCHAR2) IS
    protocol_version VARCHAR2(10);
    s                INTEGER;
    service          VARCHAR2(30);
  BEGIN
    s := dbms_pipe.receive_message('stock_service');
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        'reading pipe');
    END IF;
    dbms_pipe.unpack_message(protocol_version);
    IF protocol_version <> '1' THEN
      raise_application_error(-20000, 'Bad protocol: ' ||
        protocol_version);
    END IF;
    dbms_pipe.unpack_message(returnpipe);
    dbms_pipe.unpack_message(service);
    IF service != 'getprice' THEN
      returnerror('Service ' || service || ' not supported');
    END IF;
    dbms_pipe.unpack_message(symbol);
  END;

  PROCEDURE return_price(errormsg in VARCHAR2, price in VARCHAR2) IS
    s INTEGER;
  BEGIN
    IF errormsg is NULL THEN
      dbms_pipe.pack_message('SUCCESS');
      dbms_pipe.pack_message(price);
    ELSE
      dbms_pipe.pack_message(errormsg);
```

```
    END IF;
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:'||to_char(s)||
        ' sending on pipe');
    END IF;
  END;
END;
```

The procedure called by the client is:

```
CREATE OR REPLACE FUNCTION stock_request (symbol VARCHAR2)
    RETURN VARCHAR2 IS
  s        INTEGER;
  price    VARCHAR2(20);
  errormsg VARCHAR2(512);
BEGIN
  dbms_pipe.pack_message('1');  -- protocol version
  dbms_pipe.pack_message(dbms_pipe.unique_session_name); -- return pipe
  dbms_pipe.pack_message('getprice');
  dbms_pipe.pack_message(symbol);
  s := dbms_pipe.send_message('stock_service');
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' sending on pipe');
  END IF;
  s := dbms_pipe.receive_message(dbms_pipe.unique_session_name);
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' receiving on pipe');
  END IF;
  dbms_pipe.unpack_message(errormsg);
  IF errormsg <> 'SUCCESS' THEN
    raise_application_error(-20000, errormsg);
  END IF;
  dbms_pipe.unpack_message(price);
  RETURN price;
END;
```

You would typically only grant execute on `dbms_stock_service` to the stock service application server, and would only grant execute on `stock_request` to those users allowed to use the service.

See Also: Chapter 2, "DBMS_ALERT"

46

DBMS_PROFILER

Oracle8*i* provides a Profiler API to profile existing PL/SQL applications and to identify performance bottlenecks. You can use the collected profiler (performance) data for performance improvement or for determining code coverage for PL/SQL applications. Application developers can use code coverage data to focus their incremental testing efforts.

The profiler API is implemented as a PL/SQL package, `DBMS_PROFILER`, that provides services for collecting and persistently storing PL/SQL profiler data.

> **Note:** `DBMS_PROFILER` treats any program unit that is compiled in `NATIVE` mode as if you do not have `CREATE` privilege, that is, you will not get any output.

This chapter discusses the following topics:

Using DBMS_PROFILER

Improving application performance is an iterative process. Each iteration involves the following steps:

1. Running the application with one or more benchmark tests with profiler data collection enabled.
2. Analyzing the profiler data and identifying performance problems.
3. Fixing the problems.

The PL/SQL profiler supports this process using the concept of a "run". A run involves running the application through benchmark tests with profiler data collection enabled. You can control the beginning and the ending of a run by calling the START_PROFILER and STOP_PROFILER functions.

A typical run involves:

- Starting profiler data collection in the run.
- Executing PL/SQL code for which profiler and code coverage data is required.
- Stopping profiler data collection, which writes the collected data for the run into database tables

> **Note:** The collected profiler data is not automatically stored when the user disconnects. You must issue an explicit call to the FLUSH_DATA or the STOP_PROFILER function to store the data at the end of the session. Stopping data collection stores the collected data.

As the application executes, profiler data is collected in memory data structures that last for the duration of the run. You can call the FLUSH_DATA function at intermediate points during the run to get incremental data and to free memory for allocated profiler data structures.

Flushing the collected data involves storing collected data in database tables. The tables should already exist in the profiler user's schema. The PROFTAB.SQL script creates the tables and other data structures required for persistently storing the profiler data.

Note that running PROFTAB.SQL drops the current tables. The PROFTAB.SQL script is in the RDBMS/ADMIN directory. Some PL/SQL operations, such as the first execution of a PL/SQL unit, may involve I/O to catalog tables to load the byte code

for the PL/SQL unit being executed. Also, it may take some time executing package initialization code the first time a package procedure or function is called.

To avoid timing this overhead, "warm up" the database before collecting profile data. To do this, run the application once without gathering profiler data.

System-Wide Profiling

You can allow profiling across all users of a system, for example, to profile all users of a package, independent of who is using it. In such cases, the SYSADMIN should use a modified `PROFLOAD.SQL` script which:

- Creates the profiler tables and sequence
- Grants SELECT/INSERT/UPDATE on those tables and sequence to all users
- Defines public synonyms for the tables and sequence

Note: Do not alter the actual fields of the tables.

See Also: "FLUSH_DATA Function" on page 46-8.

Requirements

`DBMS_PROFILER` must be installed as `SYS`.

Use the `PROFLOAD.SQL` script to load the PL/SQL Profiler packages.

Collected Data

With the Probe Profiler API, you can generate profiling information for all named library units that are executed in a session. The profiler gathers information at the PL/SQL virtual machine level. This information includes the total number of times each line has been executed, the total amount of time that has been spent executing that line, and the minimum and maximum times that have been spent on a particular execution of that line.

Note: It is possible to infer the code coverage figures for PL/SQL units for which data has been collected.

The profiling information is stored in database tables. This enables querying on the data: you can build customizable reports (summary reports, hottest lines, code coverage data, and so on. And you can analyze the data.

PROFTAB.SQL

The PROFTAB.SQL script creates tables with the columns, datatypes, and definitions as shown in Table 46–1, Table 46–2, and Table 46–3.

Table 46–1 Columns in Table PLSQL_PROFILER_RUNS

Column	Datatype	Definition
runid	number primary key	Unique run identifier from plsql_profiler_ runnumber
related_run	number	Runid of related run (for client / server correlation)
run_owner	varchar2(32),	User who started run
run_date	date	Start time of run
run_comment	varchar2(2047)	User provided comment for this run
run_total_time	number	Elapsed time for this run in nanoseconds
run_system_info	varchar2(2047)	Currently unused
run_comment1	varchar2(2047)	Additional comment
spare1	varchar2(256)	Unused

Table 46–2 Columns in Table PLSQL_PROFILER_UNITS

Column	Datatype	Definition
runid	number	Primary key, references plsql_profiler_runs,
unit_number	number	Primary key, internally generated library unit #
unit_type	varchar2(32)	Library unit type
unit_owner	varchar2(32)	Library unit owner name
unit_name	varchar2(32)	Library unit name timestamp on library unit
unit_timestamp	date	In the future will be used to detect changes to unit between runs

Table 46–2 Columns in Table PLSQL_PROFILER_UNITS

Column	Datatype	Definition
total_time	number	Total time spent in this unit in nanoseconds. The profiler does not set this field, but it is provided for the convenience of analysis tools.
spare1	number	Unused
spare2	number	Unused

Table 46–3 Columns in Table PLSQL_PROFILER_DATA

Column	Datatype	Definition
runid	number	Primary key, unique (generated) run identifier
unit_number	number	Primary key, internally generated library unit number
line#	number	Primary key, not null, line number in unit
total_occur	number	Number of times line was executed
total_time	number	Total time spent executing line in nanoseconds
min_time	number	Minimum execution time for this line in nanoseconds
max_time	number	Maximum execution time for this line in nanoseconds
spare1	number	Unused
spare2	number	Unused
spare3	number	Unused
spare4	number	Unused

With Oracle8, a sample textual report writer(profrep.sql) is provided with the PL/SQL demo scripts.

Security

The profiler only gathers data for units for which a user has CREATE privilege; you cannot use the package to profile units for which EXECUTE ONLY access has been granted. In general, if a user can debug a unit, the same user can profile it. However, a unit can be profiled whether or not it has been compiled DEBUG. Oracle advises that modules that are being profiled should be compiled DEBUG, since this provides additional information about the unit in the database

Two Methods of Exception Generation

Each routine in this package has two versions that allow you to determine how errors are reported.

- A function that returns success/failure as a status value and will never raise an exception
- A procedure that returns normally if it succeeds and raises an exception if it fails

In each case, the parameters of the function and procedure are identical. Only the method by which errors are reported differs. If there is an error, there is a correspondence between the error codes that the functions return, and the exceptions that the procedures raise.

To avoid redundancy, the following section only provides details about the functional form.

Exceptions

Table 46–4 DBMS_PROFILER Exceptions

Exception	Description
version_mismatch	Corresponds to error_version.
profiler_error	Corresponds to either "error_param" or "error_io".

Error Codes

A `0` return value from any function denotes successful completion; a nonzero return value denotes an error condition. The possible errors are as follows:

- 'A subprogram was called with an incorrect parameter.'

  ```
  error_param constant binary_integer := 1;
  ```

- 'Data flush operation failed. Check whether the profiler tables have been created, are accessible, and that there is adequate space.'

  ```
  error_io     constant binary_integer := 2;
  ```

- There is a mismatch between package and database implementation. Oracle returns this error if an incorrect version of the `DBMS_PROFILER` package is installed, and if the version of the profiler package cannot work with this

database version. The only recovery is to install the correct version of the package.

```
error_version constant binary_integer := -1;
```

Summary of DBMS_PROFILER Subprograms

Table 46–5 DBMS_PROFILER Subprograms

Subprogram	Description
START_PROFILER Function on page 46-7	Starts profiler data collection in the user's session.
STOP_PROFILER Function on page 46-8	Stops profiler data collection in the user's session.
FLUSH_DATA Function on page 46-8	Flushes profiler data collected in the user's session.
PAUSE_PROFILER Function on page 46-9	Pauses profiler data collection.
RESUME_PROFILER Function on page 46-9	Resumes profiler data collection.
GET_VERSION Procedure on page 46-9	Gets the version of this API.
INTERNAL_VERSION_CHECK Function on page 46-10	Verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

START_PROFILER Function

This function starts profiler data collection in the user's session.

Syntax

There are two overloaded forms of the START_PROFILER function; one returns the run number of the started run, as well as the result of the call. The other does not return the run number. The first form is intended for use with GUI-based tools controlling the profiler.

The first form is:

```
DBMS_PROFILER.START_PROFILER(run_comment IN VARCHAR2 := sysdate,
run_comment1 IN VARCHAR2 :='',
run_number OUT BINARY_INTEGER)
RETURN BINARY_INTEGER;
```

The second form is:

```
DBMS_PROFILER.START_PROFILER(run_comment IN VARCHAR2 := sysdate,
run_comment1 IN VARCHAR2 :='')
RETURN BINARY_INTEGER;
```

Parameters

Table 46–6 START_PROFILER Function Parameters

Parameter	Description
run_comment	Each profiler run can be associated with a comment. For example, the comment could provide the name and version of the benchmark test that was used to collect data.
run_number	Stores the number of the run so you can store and later recall the run's data.
run_comment1	Allows you to make interesting comments about the run.

STOP_PROFILER Function

This function stops profiler data collection in the user's session.

This function has the side effect of flushing data collected so far in the session, and it signals the end of a run.

Syntax

```
DBMS_PROFILER.STOP_PROFILER
  RETURN BINARY_INTEGER;
```

FLUSH_DATA Function

This function flushes profiler data collected in the user's session. The data is flushed to database tables, which are expected to preexist.

Note: Use the PROFTAB.SQL script to create the tables and other data structures required for persistently storing the profiler data.

Syntax

```
DBMS_PROFILER.FLUSH_DATA
  RETURN BINARY_INTEGER;
```

PAUSE_PROFILER Function

This function pauses profiler data collection.

RESUME_PROFILER Function

This function resumes profiler data collection.

GET_VERSION Procedure

This procedure gets the version of this API.

Syntax

```
DBMS_PROFILER.GET_VERSION (
   major  OUT BINARY_INTEGER,
   minor  OUT BINARY_INTEGER);
```

Parameters

Table 46–7 GET_VERSION Procedure Parameters

Parameter	Description
major	Major version of DBMS_PROFILER.
minor	Minor version of DBMS_PROFILER.

INTERNAL_VERSION_CHECK Function

This function verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

Syntax

```
DBMS_PROFILER.INTERNAL_VERSION_CHECK
  RETURN BINARY_INTEGER;
```

47

DBMS_PROPAGATION_ADM

The DBMS_PROPAGATION_ADM package provides administrative procedures for configuring propagation from a source queue to a destination queue.

This chapter contains the following topic:

- Summary of DBMS_PROPAGATION_ADM Subprograms

> **See Also:** *Oracle9i Streams* for more information about propagation in a Streams environment

Summary of DBMS_PROPAGATION_ADM Subprograms

Table 47–1 DBMS_PROPAGATION_ADM Subprograms

Subprogram	Description
"ALTER_PROPAGATION Procedure" on page 47-3	Adds, alters, or removes a rule set for a propagation job
"CREATE_PROPAGATION Procedure" on page 47-4	Creates a propagation job and specifies the source queue, destination queue, and rule set for the propagation job.
"DROP_PROPAGATION Procedure" on page 47-7	Drops a propagation job

Note: All procedures commit unless specified otherwise.

ALTER_PROPAGATION Procedure

Adds, alters, or removes a rule set for a propagation job.

> **See Also:** *Oracle9i Streams* and Chapter 64, "DBMS_RULE_ADM" for more information about rules and rule sets

Syntax

```
DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
   propagation_name   IN VARCHAR2,
   rule_set_name      IN VARCHAR2);
```

Parameters

Table 47–2 ALTER_PROPAGATION Procedure Parameters

Parameter	Description
`propagation_name`	The name of the propagation job being altered. You must specify an existing propagation job name.
`rule_set_name`	The name of the rule set that contains the propagation rules for this propagation job. If you want to use a rule set for the propagation job, then you must specify an existing rule set in the form `[`*`schema_name`*`.]`*`rule_set_name`*. For example, to specify a rule set in the `hr` schema named `prop_rules`, enter `hr.prop_rules`. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the `DBMS_RULE_ADM` package. If you specify `NULL`, then the propagation job propagates all LCRs and user messages in its queue.

CREATE_PROPAGATION Procedure

Creates a propagation job and specifies the source queue, destination queue, and any rule set for the propagation job. A propagation job propagates events in a local source queue to a destination queue. The destination queue may or may not be in the same database as the source queue. The user who runs this procedure owns the propagation job.

This procedure also starts propagation and establishes a default schedule for the propagation job. The default schedule has the following properties:

- The start time is `SYSDATE()`.
- The duration is `NULL`, which means infinite.
- The next time is `NULL`, which means that propagation restarts as soon as it finishes the current duration.
- The latency is five seconds, which is the wait time for a message to be propagated to a destination queue after it is enqueued into a queue with no messages requiring propagation to the same destination queue.

After the propagation job is created, you can administer it using the following procedures in the `DBMS_AQADM` package:

- To alter the default schedule for a propagation job, use the `ALTER_PROPAGATION_SCHEDULE` procedure.
- To stop propagation, use the `DISABLE_PROPAGATION_SCHEDULE` procedure and specify the source queue for the `queue_name` parameter and the database link for the `destination` parameter.
- To restart propagation, use the `ENABLE_PROPAGATION_SCHEDULE` procedure and specify the source queue for the `queue_name` parameter and the database link for the `destination` parameter. Restarting propagation may be necessary if a propagation job is disabled automatically due to errors.

These types of changes affect all propagation jobs on the database link for the source queue.

The user who owns the source queue is the user who propagates events. This user must have the necessary privileges to propagate events. These privileges include the following:

- Execute privilege on the rule set used by the propagation job
- Execute privilege on all transformation functions used in the rule set
- Enqueue privilege on the destination queue if the destination queue is in the same database

If the propagation job propagates events to a destination queue in a remote database, then the owner of the source queue must be able to use the propagation job's database link and the user to which the database link connects at the remote database must have enqueue privilege on the destination queue.

Note:

- Currently, a single propagation job propagates all events that use a particular database link, even if the database link propagates events to multiple destination queues.
- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users may or may not be the same.

See Also:

- Chapter 64, "DBMS_RULE_ADM"
- *Oracle9i Streams*

Syntax

```
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
   propagation_name         IN VARCHAR2,
   source_queue             IN VARCHAR2,
   destination_queue        IN VARCHAR2,
   destination_dblink       IN VARCHAR2  DEFAULT NULL,
   rule_set_name            IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 47–3 CREATE_PROPAGATION Procedure Parameters

Parameter	Description
propagation_name	The name of the propagation job being created. A NULL setting is not allowed.
source_queue	The name of the source queue. The current database must contain the source queue.
destination_queue	The name of the destination queue
destination_dblink	The name of the database link that will be used by the propagation job. The database link is from the database that contains the source queue to the database that contains the destination queue. If NULL, then the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
rule_set_name	The name of the rule set that contains the propagation rules for this propagation job. You must specify an existing rule set in the form [*schema_name.*]*rule_set_name*. For example, to specify a rule set in the hr schema named prop_rules, enter hr.prop_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_RULE_ADM package. If you specify NULL, then the propagation job propagates all LCRs and user messages in its queue.

DROP_PROPAGATION Procedure

Drops a propagation job and deletes all messages for the destination queue in the source queue. This procedure also removes the schedule for propagation from the source queue to the destination queue.

Syntax

```
DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
   propagation_name      IN VARCHAR2);
```

Parameter

Table 47–4 DROP_PROPAGATION Procedure Parameter

Parameter	Description
propagation_name	The name of the propagation job being dropped. You must specify an existing propagation job name.

48

DBMS_RANDOM

The DBMS_RANDOM package provides a built-in random number generator. It is faster than generators written in PL/SQL because it calls Oracle's internal random number generator.

This chapter discusses the following topics:

- Requirements
- Summary of DBMS_RANDOM Subprograms

Requirements

DBMS_RANDOM must be initialized before calling the random number generator. The generator produces 8-digit integers. If the initialization subprogram is not called, then the package raises an exception.

Summary of DBMS_RANDOM Subprograms

Table 48–1 DBMS_RANDOM Package Subprograms

Subprogram	Description
INITIALIZE Procedure on page 48-2	Initializes the package with a seed value.
SEED Procedure on page 48-3	Resets the seed.
RANDOM Function on page 48-3	Gets the random number.
TERMINATE Procedure on page 48-3	Closes the package.

INITIALIZE Procedure

To use the package, first call the initialize subprogram with the seed to use.

Syntax

```
DBMS_RANDOM.INITIALIZE (
   seed  IN  BINARY_INTEGER);
```

Note: Use a seed that is sufficiently large, more than 5 digits. A single digit may not return sufficiently random numbers. Also consider getting the seed from variable values such as the time.

Parameters

Table 48–2 INITIALIZE Procedure Parameters

Parameter	Description
seed	Seed number used to generate a random number.

SEED Procedure

This procedure resets the seed.

Syntax

```
DBMS_RANDOM.SEED (
   seed  IN  BINARY_INTEGER);
```

Parameters

Table 48–3 INITIALIZE Procedure Parameters

Parameter	Description
seed	Seed number used to generate a random number.

RANDOM Function

This function gets the random number.

Syntax

```
DBMS_RANDOM.RANDOM
   RETURN BINARY_INTEGER;
```

TERMINATE Procedure

When you are finished with the package, call the TERMINATE procedure.

Syntax

```
DBMS_RANDOM.TERMINATE;
```

49

DBMS_RECTIFIER_DIFF

The DBMS_RECTIFIER_DIFF package contains APIs used to detect and resolve data inconsistencies between two replicated sites.

This chapter discusses the following topics:

Summary of DBMS_RECTIFIER_DIFF Subprograms

Table 49–1 DBMS_RECTIFIER_DIFF Package Subprograms

Subprogram	Description
DIFFERENCES Procedure on page 49-2	Determines the differences between two tables.
RECTIFY Procedure on page 49-5	Resolves the differences between two tables.

DIFFERENCES Procedure

This procedure determines the differences between two tables. It accepts the storage table of a nested table.

Note: This procedure cannot be used on LOB columns, nor on columns based on user-defined types.

Syntax

```
DBMS_RECTIFIER_DIFF.DIFFERENCES (
   sname1               IN  VARCHAR2,
   oname1               IN  VARCHAR2,
   reference_site       IN  VARCHAR2 := '',
   sname2               IN  VARCHAR2,
   oname2               IN  VARCHAR2,
   comparison_site      IN  VARCHAR2 := '',
   where_clause         IN  VARCHAR2 := '',
   { column_list        IN  VARCHAR2 := '',
   | array_columns      IN  dbms_utility.name_array, }
   missing_rows_sname   IN  VARCHAR2,
   missing_rows_oname1  IN  VARCHAR2,
   missing_rows_oname2  IN  VARCHAR2,
   missing_rows_site    IN  VARCHAR2 := '',
   max_missing          IN  INTEGER,
   commit_rows          IN  INTEGER := 500);
```

Note: This procedure is overloaded. The `column_list` and `array_columns` parameters are mutually exclusive.

Parameters

Table 49–2 DIFFERENCES Procedure Parameters **(Page 1 of 2)**

Parameter	Description
`sname1`	Name of the schema at `reference_site`.
`oname1`	Name of the table at `reference_site`.
`reference_site`	Name of the reference database site. The default, `NULL`, indicates the current site.
`sname2`	Name of the schema at `comparison_site`.
`oname2`	Name of the table at `comparison_site`.
`comparison_site`	Name of the comparison database site. The default, `NULL`, indicates the current site.
`where_clause`	Only rows satisfying this clause are selected for comparison. The default, `NULL`, indicates all rows are compared.
`column_list`	A comma-delimited list of one or more column names being compared for the two tables. You must not have any spaces before or after a comma. The default, `NULL`, indicates that all columns will be compared.
`array_columns`	A PL/SQL index-by table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be `NULL`. If position 1 is `NULL`, then all columns are used.
`missing_rows_sname`	Name of the schema containing the tables with the missing rows.
`missing_rows_oname1`	Name of an existing table at `missing_rows_site` that stores information about the rows in the table at `reference_site` that are missing from the table at `comparison_site`, and information about the rows at `comparison_site` site that are missing from the table at `reference_site`.

Table 49–2 DIFFERENCES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
missing_rows_oname2	Name of an existing table at missing_rows_site that stores information about the missing rows. This table has three columns: the R_ID column shows the rowid of the row in the missing_rows_oname1 table, the PRESENT column shows the name of the site where the row is present, and the ABSENT column shows name of the site from which the row is absent.
missing_rows_site	Name of the site where the missing_rows_oname1 and missing_rows_oname2 tables are located. The default, NULL, indicates that the tables are located at the current site.
max_missing	Integer that specifies the maximum number of rows that should be inserted into the missing_rows_oname table. If more than max_missing rows are missing, then that many rows are inserted into missing_rows_oname, and the routine then returns normally without determining whether more rows are missing. This parameter is useful if the fragments are so different that the missing rows table has too many entries and there is no point in continuing. Raises exception badnumber if max_missing is less than 1 or NULL.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should be issued only after all rows for a single table have been inserted or deleted.

Exceptions

Table 49–3 DIFFERENCES Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	The commit_rows parameter is less than 1.
missingprimarykey	Column list must include primary key (or SET_COLUMNS equivalent).
badname	NULL or empty string for table or schema name.
cannotbenull	Parameter cannot be NULL.
notshapeequivalent	Tables being compared are not shape equivalent. Shape refers to the number of columns, their column names, and the column datatypes.
unknowncolumn	Column does not exist.
unsupportedtype	Type not supported.
dbms_repcat.commfailure	Remote site is inaccessible.
dbms_ repcat.missingobject	Table does not exist.

Restrictions

The error ORA-00001 (unique constraint violated) is issued when there are any unique or primary key constraints on the missing rows table.

RECTIFY Procedure

This procedure resolves the differences between two tables. It accepts the storage table of a nested table.

> **Note:** This procedure cannot be used on LOB columns, nor on columns based on user-defined types.

Syntax

```
DBMS_RECTIFIER_DIFF.RECTIFY (
   sname1              IN  VARCHAR2,
```

```
oname1              IN  VARCHAR2,
reference_site      IN  VARCHAR2 := '',
sname2              IN  VARCHAR2,
oname2              IN  VARCHAR2,
comparison_site     IN  VARCHAR2 := '',
{ column_list       IN  VARCHAR2 := '',
| array_columns     IN  dbms_utility.name_array, }
missing_rows_sname  IN  VARCHAR2,
missing_rows_oname1 IN  VARCHAR2,
missing_rows_oname2 IN  VARCHAR2,
missing_rows_site   IN  VARCHAR2 := '',
commit_rows         IN  INTEGER := 500);
```

Note: This procedure is overloaded. The `column_list` and `array_columns` parameters are mutually exclusive.

Parameters

Table 49–4 RECTIFY Procedure Parameters **(Page 1 of 2)**

Parameter	Description
`sname1`	Name of the schema at `reference_site`.
`oname1`	Name of the table at `reference_site`.
`reference_site`	Name of the reference database site. The default, `NULL`, indicates the current site.
`sname2`	Name of the schema at `comparison_site`.
`oname2`	Name of the table at `comparison_site`.
`comparison_site`	Name of the comparison database site. The default, `NULL`, indicates the current site.
`column_list`	A comma-delimited list of one or more column names being compared for the two tables. You must not have any spaces before or after a comma. The default, `NULL`, indicates that all columns will be compared.
`array_columns`	A PL/SQL index-by table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be `NULL`. If position 1 is `NULL`, then all columns are used.
`missing_rows_sname`	Name of the schema containing the tables with the missing rows.

Table 49–4 RECTIFY Procedure Parameters **(Page 2 of 2)**

Parameter	Description
missing_rows_oname1	Name of the table at missing_rows_site that stores information about the rows in the table at reference_site that are missing from the table at comparison_site, and information about the rows at comparison_site that are missing from the table at reference_site.
missing_rows_oname2	Name of the table at missing_rows_site that stores information about the missing rows. This table has three columns: the rowid of the row in the missing_rows_oname1 table, the name of the site at which the row is present, and the name of the site from which the row is absent.
missing_rows_site	Name of the site where the missing_rows_oname1 and missing_rows_oname2 tables are located. The default, NULL, indicates that the tables are located at the current site.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string ('') or NULL indicates that a COMMIT should be issued only after all rows for a single table have been inserted or deleted.

Exceptions

Table 49–5 RECTIFY Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	The commit_rows parameter is less than 1.
badname	NULL or empty string for table or schema name.
dbms_repcat.commfailure	Remote site is inaccessible.
dbms_repcat.missingobject	Table does not exist.

50

DBMS_REDEFINITION

With DBMS_REDEFINITION, you can perform an online redefinition of tables. To achieve this online redefinition, incrementally maintainable local materialized views are used. Snapshot logs need to be defined on the master tables to support incrementally maintainable materialized views. These logs keep track of the changes to the master tables and are used by the materialized views during refresh synchronization. Restrictions on the tables that can be redefined online are as follows:

- Tables that have materialized views and materialized view logs defined on them cannot be redefined online.
- Tables that are materialized view container tables and AQ tables cannot be redefined online.
- The overflow table of an IOT table cannot be redefined online.

See Also: *Oracle9i Database Administrator's Guide* for more information.

This chapter discusses the following topics:

Constants for DBMS_REDEFINITION

The following constants are defined for this package:

- `cons_use_pk constant BINARY_INTEGER := 1;`
- `cons_use_rowid constant BINARY_INTEGER := 2;`

Summary of DBMS_REDEFINITION Subprograms

Table 50–1 DBMS_REDEFINITION Subprograms

Subprogram	Description
CAN_REDEF_TABLE Procedure on page 50-2	Determines if a given table can be redefined online.
START_REDEF_TABLE Procedure on page 50-3	Initiates the redefinition process.
FINISH_REDEF_TABLE Procedure on page 50-4	Completes the redefinition process.
SYNC_INTERIM_TABLE Procedure on page 50-5	Keeps the interim table synchronized with the original table.
ABORT_REDEF_TABLE Procedure on page 50-5	Cleans up errors that occur during the redefinition process.

CAN_REDEF_TABLE Procedure

This procedure determines if a given table can be redefined online. This is the first step of the online redefinition process. If the table is not a candidate for online redefinition, an error message is raised.

Syntax

```
DBMS_REDEFINITION.can_redef_table (
   uname        IN  VARCHAR2,
   tname        IN  VARCHAR2,
   options_flag IN  BINARY_INTEGER := 1);
```

Exceptions

If the table is not a candidate for online redefinition, an error message is raised.

Parameters

Table 50–2 CAN_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the table.
tname	The name of the table to be redefined.
options_flag	Indicates the type of redefinition method to use. If the value of this flag is dbms_redefinition.cons_use_pk, then the redefinition is done using primary keys. If the value of this flag is dbms_redefinition.cons_use_rowid, then the redefinition is done using rowids. The default method of redefinition is using primary keys.

START_REDEF_TABLE Procedure

This procedure initiates the redefinition process. After verifying that the table can be redefined online, you create an empty interim table (in the same schema as the table to be redefined) with the desired attributes of the post-redefinition table.

Syntax

```
DBMS_REDINITION.start_redef_table (
   uname        IN VARCHAR2,
   orig_table   IN VARCHAR2,
   int_table    IN VARCHAR2,
   col_mapping  IN VARCHAR2 := NULL,
   options_flag IN BINARY_INTEGER := 1);
```

Parameters

Table 50–3 START_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.

Table 50–3 START_REDEF_TABLE Procedure Parameters

Parameter	Description
col_mapping	The mapping information from the columns in the interim table to the columns in the original table. (This is similar to the column list on the SELECT clause of a query.) If NULL, all the columns in the original table are selected and have the same name after redefinition.
options_flag	Indicates the type of redefinition method to use. If the value of this flag is dbms_redefinition.cons_use_pk, then the redefinition is done using primary keys. If the value of this flag is dbms_redefinition.cons_use_rowid, then the redefinition is done using rowids. The default method of redefinition is using primary keys.

FINISH_REDEF_TABLE Procedure

This procedure completes the redefinition process. Before this step, you can create new indexes, triggers, grants, and constraints on the interim table. The referential constraints involving the interim table must be disabled. After completing this step, the original table is redefined with the attributes and data of the interim table. The original table is locked briefly during this procedure.

Syntax

```
DBMS_REDFINITION.finish_redef_table (
   uname       IN VARCHAR2,
   orig_table  IN VARCHAR2,
   int_table   IN VARCHAR2);
```

Parameters

Table 50–4 FINISH_REDEF_TABLE Procedure Parameters

Parameters	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.

SYNC_INTERIM_TABLE Procedure

This procedure keeps the interim table synchronized with the original table. This step is useful in minimizing the amount of synchronization needed to be done by `finish_reorg_table` before completing the online redefinition. This procedure can be called between long running operations (such as create index) on the interim table to sync it up with the data in the original table and speed up subsequent operations.

Syntax

```
DBMS_REDFINITION.sync_interim_table (
   uname       IN VARCHAR2,
   orig_table  IN  VARCHAR2,
   int_table   IN VARCHAR2);
```

Parameters

Table 50–5 SYNC_INTERIM_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the table.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.

ABORT_REDEF_TABLE Procedure

This procedure cleans up errors that occur during the redefinition process. This procedure can also be used to abort the redefinition process any time after `start_reorg_table` has been called and before `finish_reorg_table` is called.

Syntax

```
DBMS_REDEFINITION.abort_redef_table (
   uname       IN VARCHAR2,
   orig_table  IN VARCHAR2,
   int_table   IN VARCHAR2);
```

Parameters

Table 50–6 ABORT_REDEF_TABLE Procedure Parameters

Parameter	Description
uname	The schema name of the tables.
orig_table	The name of the table to be redefined.
int_table	The name of the interim table.

51

DBMS_REFRESH

DBMS_REFRESH enables you to create groups of materialized views that can be refreshed together to a transactionally consistent point in time.

This chapter discusses the following topics:

Summary of DBMS_REFRESH Subprograms

Table 51–1 DBMS_REFRESH Package Subprograms

Subprogram	Description
ADD Procedure on page 51-2	Adds materialized views to a refresh group.
CHANGE Procedure on page 51-3	Changes the refresh interval for a refresh group.
DESTROY Procedure on page 51-5	Removes all of the materialized views from a refresh group and deletes the refresh group.
MAKE Procedure on page 51-6	Specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.
REFRESH Procedure on page 51-8	Manually refreshes a refresh group.
SUBTRACT Procedure on page 51-9	Removes materialized views from a refresh group.

ADD Procedure

This procedure adds materialized views to a refresh group.

See Also: *Oracle9i Replication* for more information

Syntax

```
DBMS_REFRESH.ADD (
   name     IN VARCHAR2,
   { list   IN VARCHAR2,
   | tab    IN DBMS_UTILITY.UNCL_ARRAY, }
   lax      IN BOOLEAN := false);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 51–2 ADD Procedures Parameters

Parameter	Description
`name`	Name of the refresh group to which you want to add members.
`list`	Comma-delimited list of materialized views that you want to add to the refresh group. (Synonyms are not supported.)
`tab`	Instead of a comma-delimited list, you can supply a PL/SQL index-by table of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a materialized view. The first materialized view should be in position 1. The last position must be `NULL`.
`lax`	A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from one group to another, then you must set the `lax` flag to `true` to succeed. Oracle then automatically removes the materialized view from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to `ADD` generates an error message.

CHANGE Procedure

This procedure changes the refresh interval for a refresh group.

See Also: *Oracle9i Replication* for more information about refresh groups

Syntax

```
DBMS_REFRESH.CHANGE (
   name                 IN VARCHAR2,
   next_date            IN DATE           := NULL,
   interval             IN VARCHAR2       := NULL,
   implicit_destroy     IN BOOLEAN        := NULL,
   rollback_seg         IN VARCHAR2       := NULL,
   push_deferred_rpc    IN BOOLEAN        := NULL,
   refresh_after_errors IN BOOLEAN        := NULL,
   purge_option         IN BINARY_INTEGER := NULL,
```

```
   parallelism           IN BINARY_INTEGER := NULL,
   heap_size             IN BINARY_INTEGER := NULL);
```

Parameters

Table 51–3 CHANGE Procedures Parameters **(Page 1 of 2)**

Parameter	Description
`name`	Name of the refresh group for which you want to alter the refresh interval.
`next_date`	Next date that you want a refresh to occur. By default, this date remains unchanged.
`interval`	Function used to calculate the next time to refresh the materialized views in the refresh group. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh. By default, the interval remains unchanged.
`implicit_destroy`	Allows you to reset the value of the `implicit_destroy` flag. If this flag is set, then Oracle automatically deletes the group if it no longer contains any members. By default, this flag remains unchanged.
`rollback_seg`	Allows you to change the rollback segment used. By default, the rollback segment remains unchanged. To reset this parameter to use the default rollback segment, specify `NULL`, including the quotes. Specifying `NULL` without quotes indicates that you do not want to change the rollback segment currently being used.
`push_deferred_rpc`	Used by updatable materialized views only. Set this parameter to `true` if you want to push changes from the materialized view to its associated master table or master materialized view before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost. By default, this flag remains unchanged.
`refresh_after_errors`	Used by updatable materialized views only. Set this parameter to `true` if you want the refresh to proceed even if there are outstanding conflicts logged in the `DEFERROR` view for the materialized view's master table or master materialized view. By default, this flag remains unchanged.

Table 51–3 CHANGE Procedures Parameters **(Page 2 of 2)**

Parameter	Description
purge_option	If you are using the parallel propagation mechanism (that is, parallelism is set to 1 or greater), then: ■ 0 = do not purge ■ 1 = lazy (default) ■ 2 = aggressive In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set purge to *do not purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

DESTROY Procedure

This procedure removes all of the materialized views from a refresh group and delete the refresh group.

See Also: *Oracle9i Replication* for more information refresh groups

Syntax

```
DBMS_REFRESH.DESTROY (
   name   IN   VARCHAR2);
```

Parameters

Table 51–4 DESTROY Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to destroy.

MAKE Procedure

This procedure specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.

See Also: *Oracle9i Replication* for more information

Syntax

```
DBMS_REFRESH.MAKE (
   name                 IN    VARCHAR2
   { list               IN    VARCHAR2,
   | tab                IN     DBMS_UTILITY.UNCL_ARRAY,}
   next_date            IN     DATE,
   interval             IN     VARCHAR2,
   implicit_destroy     IN      BOOLEAN          := false,
   lax                  IN      BOOLEAN          := false,
   job                  IN      BINARY INTEGER   := 0,
   rollback_seg         IN      VARCHAR2         := NULL,
   push_deferred_rpc    IN      BOOLEAN          := true,
   refresh_after_errors IN      BOOLEAN          := false)
   purge_option          IN     BINARY_INTEGER := NULL,
   parallelism           IN     BINARY_INTEGER := NULL,
   heap_size             IN     BINARY_INTEGER := NULL);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 51–5 MAKE Procedure Parameters **(Page 1 of 2)**

Parameter	Description
name	Unique name used to identify the refresh group. Refresh groups must follow the same naming conventions as tables.
list	Comma-delimited list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database.
tab	Instead of a comma separated list, you can supply a PL/SQL index-by table of names of materialized views that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of *n* materialized views, then the first materialized view should be in position 1 and the *n* + 1 position should be set to NULL.
next_date	Next date that you want a refresh to occur.
interval	Function used to calculate the next time to refresh the materialized views in the group. This field is used with the next_date value. For example, if you specify NEXT_DAY(SYSDATE+1, "MONDAY") as your interval, and if your next_date evaluates to Monday, then Oracle refreshes the materialized views every Monday. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh.
implicit_destroy	Set this to true if you want to delete the refresh group automatically when it no longer contains any members. Oracle checks this flag only when you call the SUBTRACT procedure. That is, setting this flag still enables you to create an empty refresh group.
lax	A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from an existing group to a new refresh group, then you must set this to true to succeed. Oracle then automatically removes the materialized view from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to MAKE generates an error message.
job	Needed by the Import utility. Use the default value, 0.
rollback_seg	Name of the rollback segment to use while refreshing materialized views. The default, NULL, uses the default rollback segment.

Table 51–5 MAKE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
push_deferred_rpc	Used by updatable materialized views only. Use the default value, true, if you want to push changes from the materialized view to its associated master table or master materialized view before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	Used by updatable materialized views only. Set this to 0 if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 = do not purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set purge to *do not purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

REFRESH Procedure

This procedure manually refreshes a refresh group.

> **See Also:** *Oracle9i Replication* for more information about refresh groups

Syntax

```
DBMS_REFRESH.REFRESH (
```

```
   name   IN    VARCHAR2);
```

Parameters

Table 51–6 REFRESH Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to refresh manually.

SUBTRACT Procedure

This procedure removes materialized views from a refresh group.

> **See Also:** *Oracle9i Replication* for more information about refresh groups

Syntax

```
DBMS_REFRESH.SUBTRACT (
   name       IN    VARCHAR2,
   { list     IN    VARCHAR2,
   | tab      IN     DBMS_UTILITY.UNCL_ARRAY, }
   lax        IN     BOOLEAN := false);
```

> **Note:** This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 51–7 SUBTRACT Procedure Parameters

Parameter	Description
`name`	Name of the refresh group from which you want to remove members.
`list`	Comma-delimited list of materialized views that you want to remove from the refresh group. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database.
`tab`	Instead of a comma-delimited list, you can supply a PL/SQL index-by table of names of materialized views that you want to refresh using the datatype `DBMS_UTILITY.UNCL_ARRAY`. If the table contains the names of *n* materialized views, then the first materialized view should be in position 1 and the *n* + 1 position should be set to `NULL`.
`lax`	Set this to `false` if you want Oracle to generate an error message if the materialized view you are attempting to remove is not a member of the refresh group.

52

DBMS_REPAIR

DBMS_REPAIR contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes. You can address corruptions where possible and continue to use objects while you attempt to rebuild or repair them.

Note: The DBMS_REPAIR package is intended for use by database administrators only. It is not intended for use by application developers.

See Also: For detailed information about using the DBMS_REPAIR package, see *Oracle9i Database Administrator's Guide*.

This chapter discusses the following topics:

- Security, Enumeration Types, and Exceptions
- Summary of DBMS_REPAIR Subprograms

Security, Enumeration Types, and Exceptions

Security

The package is owned by SYS. Execution privilege is not granted to other users.

Enumeration Types

The DBMS_REPAIR package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS_REPAIR.TABLE_OBJECT.

Table 52–1 lists the parameters and the enumerated constants.

Table 52–1 DBMS_REPAIR Enumeration Types

Parameter	Constant
object_type	TABLE_OBJECT, INDEX_OBJECT, CLUSTER_OBJECT
action	CREATE_ACTION, DROP_ACTION, PURGE_ACTION
table_type	REPAIR_TABLE, ORPHAN_TABLE
flags	SKIP_FLAG, NOSKIP_FLAG

Note: The default table_name will be REPAIR_TABLE when table_type is REPAIR_TABLE, and will be ORPHAN_KEY_TABLE when table_type is ORPHAN_TABLE.

Exceptions

Table 52–2 DBMS_REPAIR Exceptions

Exception	Description	Action
942	Reported by DBMS_REPAIR.ADMIN_TABLES during a DROP_ACTION when the specified table doesn't exist.	
955	Reported by DBMS_REPAIR.CREATE_ACTION when the specified table already exists.	

Table 52–2 DBMS_REPAIR Exceptions

Exception	Description	Action
24120	An invalid parameter was passed to the specified DBMS_REPAIR procedure.	Specify a valid parameter value or use the parameter's default.
24122	An incorrect block range was specified.	Specify correct values for the BLOCK_START and BLOCK_END parameters.
24123	An attempt was made to use the specified feature, but the feature is not yet implemented.	Do not attempt to use the feature.
24124	An invalid ACTION parameter was specified.	Specify CREATE_ACTION, PURGE_ACTION or DROP_ACTION for the ACTION parameter.
24125	An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since DBMS_REPAIR.CHECK_OBJECT was run.	Use DBMS_REPAIR.ADMIN_TABLES to purge the repair table and run DBMS_REPAIR.CHECK_OBJECT to determine whether there are any corrupt blocks to be fixed.
24127	TABLESPACE parameter specified with an ACTION other than CREATE_ACTION.	Do not specify TABLESPACE when performing actions other than CREATE_ACTION.
24128	A partition name was specified for an object that is not partitioned.	Specify a partition name only if the object is partitioned.
24129	An attempt was made to pass a table name parameter without the specified prefix.	Pass a valid table name parameter.
24130	An attempt was made to specify a repair or orphan table that does not exist.	Specify a valid table name parameter.
24131	An attempt was made to specify a repair or orphan table that does not have a correct definition.	Specify a table name that refers to a properly created table.
24132	An attempt was made to specify a table name is greater than 30 characters long.	Specify a valid table name parameter.

Summary of DBMS_REPAIR Subprograms

Table 52–3 DBMS_REPAIR Package Subprograms

Subprogram	Description
ADMIN_TABLES Procedure on page 52-4	Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions.
CHECK_OBJECT Procedure on page 52-5	Detects and reports corruptions in a table or index.
DUMP_ORPHAN_KEYS Procedure on page 52-7	Reports on index entries that point to rows in corrupt data blocks.
FIX_CORRUPT_BLOCKS Procedure on page 52-8	Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT.
REBUILD_FREELISTS Procedure on page 52-9	Rebuilds an object's freelists.
SKIP_CORRUPT_BLOCKS Procedure on page 52-10	Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered.
SEGMENT_FIX_STATUS Procedure on page 52-11	Fixes the corrupted state of a bitmap entry.

ADMIN_TABLES Procedure

This procedure provides administrative functions for the DBMS_REPAIR package repair and orphan key tables.

Syntax

```
DBMS_REPAIR.ADMIN_TABLES (
   table_name  IN   VARCHAR2,
   table_type  IN   BINARY_INTEGER,
   action      IN   BINARY_INTEGER,
   tablespace  IN   VARCHAR2        DEFAULT NULL);
```

Parameters

Table 52–4 ADMIN_TABLES Procedure Parameters

Parameter	Description
table_name	Name of the table to be processed. Defaults to ORPHAN_KEY_TABLE or REPAIR_TABLE based on the specified table_type. When specified, the table name must have the appropriate prefix: ORPHAN_ or REPAIR_.
table_type	Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE. See "Enumeration Types" on page 52-2.
action	Indicates what administrative action to perform. Must be either CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and if CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and if DROP_ACTION is specified, then an error is returned. When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA_<table_name> is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated. Created in the SYS schema. See "Enumeration Types" on page 52-2.
tablespace	Indicates the tablespace to use when creating a table. By default, the SYS default tablespace is used. An error is returned if the tablespace is specified and if the action is not CREATE_ACTION.

CHECK_OBJECT Procedure

This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.

Validation consists of block checking all blocks in the object. You may optionally specify a DBA range, partition name, or subpartition name when you want to check a portion of an object.

Syntax

```
DBMS_REPAIR.CHECK_OBJECT (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   relative_fno      IN  BINARY_INTEGER DEFAULT NULL,
   block_start       IN  BINARY_INTEGER DEFAULT NULL,
   block_end         IN  BINARY_INTEGER DEFAULT NULL,
   corrupt_count     OUT BINARY_INTEGER);
```

Parameters

Table 52–5 CHECK_OBJECT Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be checked.
object_name	Name of the table or index to be checked.
partition_name	Partition or subpartition name to be checked. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are checked. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are checked.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Enumeration Types" on page 52-2.
repair_table_name	Name of the repair table to be populated. The table must exist in the SYS schema. Use the admin_tables procedure to create a repair table. The default name is REPAIR_TABLE.
flags	Reserved for future use.
relative_fno	Relative file number: Used when specifying a block range.
block_start	First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.

Table 52–5 CHECK_OBJECT Procedure Parameters

Parameter	Description
block_end	Last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of block_start or block_end is specified, then the other defaults to the first or last block in the file respectively.
corrupt_count	Number of corruptions reported.

DUMP_ORPHAN_KEYS Procedure

This procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

Syntax

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT INDEX_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   orphan_table_name IN  VARCHAR2       DEFAULT 'ORPHAN_KEYS_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   key_count         OUT BINARY_INTEGER);
```

Parameters

Table 52–6 DUMP_ORPHAN_KEYS Procedure Parameters

Parameter	Description
schema_name	Schema name.

Table 52–6 DUMP_ORPHAN_KEYS Procedure Parameters

Parameter	Description
object_name	Object name.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. The default is INDEX_OBJECT See "Enumeration Types" on page 52-2.
repair_table_name	Name of the repair table that has information regarding corrupt blocks in the base table. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
orphan_table_name	Name of the orphan key table to populate with information regarding each index entry that refers to a row in a corrupt data block. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
flags	Reserved for future use.
key_count	Number of index entries processed.

FIX_CORRUPT_BLOCKS Procedure

This procedure fixes the corrupt blocks in specified objects based on information in the repair table that was previously generated by the check_object procedure.

Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is effected, the associated row in the repair table is updated with a fix timestamp.

Syntax

```
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
```

```
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   fix_count         OUT BINARY_INTEGER);
```

Parameters

Table 52–7 FIX_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object with corrupt blocks to be fixed.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Enumeration Types" on page 52-2.
repair_table_name	Name of the repair table with the repair directives. Must exist in the SYS schema.
flags	Reserved for future use.
fix_count	Number of blocks fixed.

REBUILD_FREELISTS Procedure

This procedure rebuilds the freelists for the specified object. All free blocks are placed on the master freelist. All other freelists are zeroed.

If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

Syntax

```
DBMS_REPAIR.REBUILD_FREELISTS (
   schema_name    IN VARCHAR2,
   partition_name IN VARCHAR2       DEFAULT NULL,
```

```
   object_type     IN BINARY_INTEGER DEFAULT TABLE_OBJECT);
```

Parameters

Table 52–8 REBUILD_FREELISTS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object whose freelists are to be rebuilt.
partition_name	Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Enumeration Types" on page 52-2.

SKIP_CORRUPT_BLOCKS Procedure

This procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object.

When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

> **Note:** When Oracle performs an index range scan on a corrupt index after DBMS_REPAIR.SKIP_CORRUPT_BLOCKS has been set for the base table, corrupt branch blocks and root blocks are not skipped. Only corrupt non-root leaf blocks are skipped.

Syntax

```
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
   schema_name  IN VARCHAR2,
   object_name  IN VARCHAR2,
   object_type  IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
   flags        IN BINARY_INTEGER DEFAULT SKIP_FLAG);
```

Parameters

Table 52–9 SKIP_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be processed.
object_name	Name of the object.
partition_name (optional)	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or CLUSTER_OBJECT. See "Enumeration Types" on page 52-2.
flags	If SKIP_FLAG is specified, then it turns on the skip of software corrupt blocks for the object during index and table scans. If NOSKIP_FLAG is specified, then scans that encounter software corrupt blocks return an ORA-1578. See "Enumeration Types" on page 52-2.

SEGMENT_FIX_STATUS Procedure

With this procedure you can fix the corrupted state of a bitmap entry. The procedure either recalculates the state based on the current contents of the corresponding block or sets the state to a specific value.

Syntax

```
DBMS_REPAIR.SEGMENT_FIX_STATUS (
   segment_owner   IN VARCHAR2,
   segment_name    IN VARCHAR2,
   segment_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
   file_number     IN BINARY_INTEGER DEFAULT NULL,
   block_number    IN BINARY_INTEGER DEFAULT NULL,
   status_value    IN BINARY_INTEGER DEFAULT NULL,
   partition_name  IN VARCHAR2 DEFAULT NULL,);
```

Parameters

Table 52–10 SEGMENT_FIX_STATUS Procedure Parameters

Parameter	Description
schema_owner	Schema name of the segment.
segment_name	Segment name.
partition_name	Optional. Name of an individual partition. NULL for nonpartitioned objects. Default is NULL.
segment_type	Optional Type of the segment (for example, TABLE or INDEX). Default is NULL.
file_number	(optional) The tablespace-relative file number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
block_number	(optional) The file-relative file number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
status_value	(optional) The value to which the block status described by the file_number and block_number will be set. If omitted, the status will be set based on the current state of the block. This is almost always the case, but if there is a bug in the calculation algorithm, the value can be set manually. Status values: 1 = block is full 2 = block is 0-25% free 3 = block is 25-50% free 4 = block is 50-75% free 5 = block is 75-100% free The status for bitmap blocks, segment headers, and extent map blocks cannot be altered. The status for blocks in a fixed hash area cannot be altered. For index blocks, there are only two possible states: 1 = block is full and 3 = block has free space.

Examples

```
/* Fix the bitmap status for all the blocks in table mytab in schema sys */
execute dbms_repair.segment_fix_status('SYS', 'MYTAB');

/* Mark block number 45, filenumber 1 for table mytab in sys schema as FULL.*/
execute dbms_repair.segment_fix_status('SYS', 'MYTAB', 1,1, 45, 1);
```

53

DBMS_REPCAT

DBMS_REPCAT provides routines to administer and update the replication catalog and environment.

This chapter discusses the following topics:

Summary of DBMS_REPCAT Subprograms

Table 53–1 DBMS_REPCAT Subprograms

Subprogram	Description
ADD_GROUPED_COLUMN Procedure on page 53-6	Adds members to an existing column group.
ADD_MASTER_DATABASE Procedure on page 53-8	Adds another master site to your replication environment.
ADD_NEW_MASTERS Procedure on page 53-10	Adds the master sites in the DBA_REPSITES_NEW data dictionary view to the replication catalog at all available master sites.
ADD_PRIORITY_datatype Procedure on page 53-16	Adds a member to a priority group.
ADD_SITE_PRIORITY_SITE Procedure on page 53-17	Adds a new site to a site priority group.
ADD_conflicttype_RESOLUTION Procedure on page 53-19	Designates a method for resolving an update, delete, or uniqueness conflict.
ALTER_CATCHUP_PARAMETERS Procedure on page 53-24	Alters the values for parameters stored in the DBA_REPEXTENSIONS data dictionary view.
ALTER_MASTER_PROPAGATION Procedure on page 53-27	Alters the propagation method for a specified replication group at a specified master site.
ALTER_MASTER_REPOBJECT Procedure on page 53-28	Alters an object in your replication environment.
ALTER_MVIEW_PROPAGATION Procedure on page 53-32	Alters the propagation method for a specified replication group at the current materialized view site.
ALTER_PRIORITY Procedure on page 53-33	Alters the priority level associated with a specified priority group member.
ALTER_PRIORITY_datatype Procedure on page 53-35	Alters the value of a member in a priority group.
ALTER_SITE_PRIORITY Procedure on page 53-36	Alters the priority level associated with a specified site.
ALTER_SITE_PRIORITY_SITE Procedure on page 53-37	Alters the site associated with a specified priority level.
CANCEL_STATISTICS Procedure on page 53-38	Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

Table 53–1 DBMS_REPCAT Subprograms

Subprogram	Description
COMMENT_ON_COLUMN_GROUP Procedure on page 53-39	Updates the comment field in the ALL_REPCOLUMN_GROUP view for a column group.
COMMENT_ON_conflicttype_RESOLUTION Procedure on page 53-46	Updates the SCHEMA_COMMENT field in the ALL_REPGROUP view for a materialized view site.
COMMENT_ON_PRIORITY_GROUP / COMMENT_ON_SITE_PRIORITY Procedures on page 53-41	Updates the comment field in the ALL_REPPRIORITY_GROUP view for a (site) priority group.
COMMENT_ON_REPGROUP Procedure on page 53-42	Updates the comment field in the ALL_REPGROUP view for a master group.
COMMENT_ON_REPOBJECT Procedure on page 53-43	Updates the comment field in the ALL_REPOBJECT view for a replicated object.
COMMENT_ON_REPSITES Procedure on page 53-44	Updates the comment field in the ALL_REPSITE view for a replicated site.
COMMENT_ON_conflicttype_RESOLUTION Procedure on page 53-46	Updates the comment field in the ALL_REPRESOLUTION view for a conflict resolution routine.
COMPARE_OLD_VALUES Procedure on page 53-47	Specifies whether to compare old column values at each master site for each nonkey column of a replicated table for updates and deletes.
CREATE_MASTER_REPGROUP Procedure on page 53-50	Creates a new, empty, quiesced master group.
CREATE_MASTER_REPOBJECT Procedure on page 53-51	Specifies that an object is a replicated object.
CREATE_MVIEW_REPGROUP Procedure on page 53-55	Creates a new, empty materialized view group in your local database.
CREATE_MVIEW_REPOBJECT Procedure on page 53-56	Adds a replicated object to a materialized view group.
DEFINE_COLUMN_GROUP Procedure on page 53-59	Creates an empty column group.
DEFINE_PRIORITY_GROUP Procedure on page 53-60	Creates a new priority group for a master group.
DEFINE_SITE_PRIORITY Procedure on page 53-61	Creates a new site priority group for a master group.

Table 53–1 DBMS_REPCAT Subprograms

Subprogram	Description
DO_DEFERRED_REPCAT_ADMIN Procedure on page 53-62	Executes the local outstanding deferred administrative procedures for the specified master group at the current master site, or for all master sites.
DROP_COLUMN_GROUP Procedure on page 53-63	Drops a column group.
DROP_GROUPED_COLUMN Procedure on page 53-64	Removes members from a column group.
DROP_MASTER_REPGROUP Procedure on page 53-65	Drops a master group from your current site.
DROP_MASTER_REPOBJECT Procedure on page 53-67	Drops a replicated object from a master group.
DROP_PRIORITY Procedure on page 53-70	Drops a replicated object from a master group.
DROP_MVIEW_REPGROUP Procedure on page 53-68	Drops a materialized view site from your replication environment.
DROP_MVIEW_REPOBJECT Procedure on page 53-69	Drops a replicated object from a materialized view site.
DROP_PRIORITY Procedure on page 53-70	Drops a member of a priority group by priority level.
DROP_PRIORITY_GROUP Procedure on page 53-71	Drops a priority group for a specified master group.
DROP_PRIORITY_datatype Procedure on page 53-72	Drops a member of a priority group by value.
DROP_SITE_PRIORITY Procedure on page 53-73	Drops a site priority group for a specified master group.
DROP_SITE_PRIORITY_SITE Procedure on page 53-74	Drops a specified site, by name, from a site priority group.
DROP_conflicttype_RESOLUTION Procedure on page 53-75	Drops an update, delete, or uniqueness conflict resolution method.
EXECUTE_DDL Procedure on page 53-77	Supplies DDL that you want to have executed at each master site.
GENERATE_MVIEW_SUPPORT Procedure on page 53-78	Activates triggers and generate packages needed to support the replication of updatable materialized views or procedural replication.

Table 53–1 DBMS_REPCAT Subprograms

Subprogram	Description
GENERATE_REPLICATION_SUPPORT Procedure on page 53-80	Generates the triggers, packages, and procedures needed to support replication for a specified object.
MAKE_COLUMN_GROUP Procedure on page 53-82	Creates a new column group with one or more members.
PREPARE_INSTANTIATED_MASTER Procedure on page 53-84	Changes the global name of the database you are adding to a master group.
PURGE_MASTER_LOG Procedure on page 53-85	Removes local messages in the DBA_REPCATLOG associated with a specified identification number, source, or master group.
PURGE_STATISTICS Procedure on page 86	Removes information from the ALL_REPRESOLUTION_STATISTICS view.
REFRESH_MVIEW_REPGROUP Procedure on page 53-87	Refreshes a materialized view group with the most recent data from its associated master site or master materialized view site.
REGISTER_MVIEW_REPGROUP Procedure on page 53-89	Facilitates the administration of materialized views at their respective master sites or master materialized view sites by inserting, modifying, or deleting from DBA_REGISTERED_MVIEW_GROUPS.
REGISTER_STATISTICS Procedure on page 53-90	Collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.
RELOCATE_MASTERDEF Procedure on page 91	Changes your master definition site to another master site in your replication environment.
REMOVE_MASTER_DATABASES Procedure on page 53-93	Removes one or more master databases from a replication environment.
RENAME_SHADOW_COLUMN_GROUP Procedure on page 53-94	Renames the shadow column group of a replicated table to make it a named column group.
REPCAT_IMPORT_CHECK Procedure on page 53-95	Ensures that the objects in the master group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by the advanced replication facility.
RESUME_MASTER_ACTIVITY Procedure on page 53-96	Resumes normal replication activity after quiescing a replication environment.

Table 53–1 DBMS_REPCAT Subprograms

Subprogram	Description
RESUME_PROPAGATION_TO_MDEF Procedure on page 53-97	Indicates that export is effectively finished and propagation for both extended and unaffected replication groups existing at master sites can be enabled.
SEND_OLD_VALUES Procedure on page 53-98	Specifies whether to send old column values for each nonkey column of a replicated table for updates and deletes.
SET_COLUMNS Procedure on page 53-100	Specifies use of an alternate column or group of columns, instead of the primary key, to determine which columns of a table to compare when using row-level replication.
SPECIFY_NEW_MASTERS Procedure on page 53-102	Specifies the master sites you intend to add to an existing replication group without quiescing the group.
SUSPEND_MASTER_ACTIVITY Procedure on page 53-105	Suspends replication activity for a master group.
SWITCH_MVIEW_MASTER Procedure on page 53-105	Changes the master site of a materialized view group to another master site.
UNDO_ADD_NEW_MASTERS_ REQUEST Procedure on page 53-107	Undoes all of the changes made by the `SPECIFY_NEW_MASTERS` and `ADD_NEW_ MASTERS` procedures for a specified `extension_id`.
UNREGISTER_MVIEW_REPGROUP Procedure on page 53-109	Facilitates the administration of materialized views at their respective master sites and master materialized view sites by inserting, modifying, or deleting from `DBA_REGISTERED_MVIEW_ GROUPS`.
VALIDATE Function on page 53-109	Validates the correctness of key conditions of a multimaster replication environment.
WAIT_MASTER_LOG Procedure on page 53-112	Determines whether changes that were asynchronously propagated to a master site have been applied.

ADD_GROUPED_COLUMN Procedure

This procedure adds members to an existing column group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2s);
```

Parameters

Table 53–2 ADD_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table with which the column group is associated. The table can be the storage table of a nested table.
column_group	Name of the column group to which you are adding members.
list_of_column_ names	Names of the columns that you are adding to the designated column group. This can either be a comma-delimited list or a PL/SQL index-by table of column names. The PL/SQL index-by table must be of type DBMS_REPCAT.VARCHAR2. Use the single value '*' to create a column group that contains all of the columns in your table. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object, then you can specify SYS_NC_OID$ to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is a storage table of a nested table, then you can specify NESTED_TABLE_ID to add the column that tracks the identifier for each row of the nested table.

Table 53–3 ADD_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
missinggroup	Specified column group does not exist.
missingcolumn	Specified column does not exist in the specified table.
duplicatecolumn	Specified column is already a member of another column group.
missingschema	Specified schema does not exist.
notquiesced	Replication group to which the specified table belongs is not quiesced.

ADD_MASTER_DATABASE Procedure

This procedure adds another master site to your replication environment. This procedure regenerates all the triggers and their associated packages at existing master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_MASTER_DATABASE (
   gname                IN   VARCHAR2,
   master               IN   VARCHAR2,
   use_existing_objects IN   BOOLEAN  := true,
   copy_rows            IN   BOOLEAN  := true,
   comment              IN   VARCHAR2 := '',
   propagation_mode     IN   VARCHAR2 := 'ASYNCHRONOUS',
   fname                IN   VARCHAR2 := NULL);
```

Parameters

Table 53–4 ADD_MASTER_DATABASE Procedure Parameters

Parameter	Description
gname	Name of the replication group being replicated. This replication group must already exist at the master definition site.
master	Fully qualified database name of the new master database.
use_existing_objects	Indicate true if you want to reuse any objects of the same type and shape that already exist in the schema at the new master site.
copy_rows	Indicate true if you want the initial contents of a table at the new master site to match the contents of the table at the master definition site.
comment	This comment is added to the MASTER_COMMENT field of the DBA_REPSITES view.
propagation_mode	Method of forwarding changes to and receiving changes from new master database. Accepted values are synchronous and asynchronous.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 53–5 ADD_MASTER_DATABASE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replication has not been suspended for the master group.
missingrepgroup	Replication group does not exist at the specified database site.
commfailure	New master is not accessible.
typefailure	An incorrect propagation mode was specified.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
duplrepgrp	Master site already exists.

ADD_NEW_MASTERS Procedure

This procedure adds the master sites in the DBA_REPSITES_NEW data dictionary view to the master groups specified when the SPECIFY_NEW_MASTERS procedure was run. Information about these new master sites are added to the replication catalog at all available master sites.

All master sites instantiated with object-level export/import must be accessible at this time. Their new replication groups are added in the quiesced state. Master sites instantiated through full database export/import or through changed-based recovery do not need to be accessible.

Run this procedure after you run the SPECIFY_NEW_MASTERS procedure.

Caution: After running this procedure, do not disable or enable propagation of the deferred transactions queue until after the new master sites are added. The DBA_REPEXTENSIONS data dictionary view must be clear before you disable or enable propagation. You can use the Replication Management tool or the SET_DISABLED procedure in the DBMS_DEFER_SYS package to disable or enable propagation.

See Also: "SPECIFY_NEW_MASTERS Procedure" on page 53-102

Syntax

```
DBMS_REPCAT.ADD_NEW_MASTERS (
   export_required                IN    BOOLEAN,
   { available_master_list        IN    VARCHAR2,
    | available_master_table       IN    DBMS_UTILITY.DBLINK_ARRAY,}
   masterdef_flashback_scn        OUT   NUMBER,
   extension_id                   OUT   RAW,
   break_trans_to_masterdef       IN    BOOLEAN   := false,
   break_trans_to_new_masters     IN    BOOLEAN   := false,
   percentage_for_catchup_mdef    IN    BINARY_INTEGER  := 100,
   cycle_seconds_mdef             IN    BINARY_INTEGER  := 60,
   percentage_for_catchup_new     IN    BINARY_INTEGER  := 100,
   cycle_seconds_new              IN    BINARY_INTEGER  := 60);
```

Note: This procedure is overloaded. The `available_master_list` and `available_master_table` parameters are mutually exclusive.

Parameters

Table 53–6 ADD_NEW_MASTERS Procedure Parameters

Parameter	Description
export_required	Set to true if either object-level or full database export is required for at least one of the new master sites. Set to false if you are using change-based recovery for all of the new master sites.
available_master_list	A comma-delimited list of the new master sites to be instantiated using object-level export/import. The sites listed must match the sites specified in the SPECIFY_NEW_MASTERS procedure. List only the new master sites, not the existing master sites. Do not put any spaces between site names. Specify NULL if all masters will be instantiated using full database export/import or change-based recovery.
available_master_table	A table that lists the new master sites to be instantiated using object-level export/import. The sites in the table must match the sites specified in the SPECIFY_NEW_MASTERS procedure. Do not specify masters that will be instantiated using full database export/import or change-based recovery. In the table that lists the master sites to be instantiated using object-level export/import, list only the new master sites for the master groups being extended. Do not list the existing master sites in the master groups being extended. The first master site should be at position 1, the second at position 2, and so on.
masterdef_flashback_scn	This OUT parameter returns a system change number (SCN) that must be used during export or change-based recovery. Use the value returned by this parameter for the FLASHBACK_SCN export parameter when you perform the export. You can find the flashback_scn value by querying the DBA_REPEXTENSIONS data dictionary view.
extension_id	This OUT parameter returns an identifier for the current pending request to add master databases without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

Table 53–6 ADD_NEW_MASTERS Procedure Parameters

Parameter	Description
break_trans_to_masterdef	This parameter is meaningful only if export_required is set to true. If break_trans_to_masterdef is set to true, then existing masters may continue to propagate their deferred transactions to the master definition site for replication groups that are not adding master sites. Deferred transactions for replication groups that are adding master sites cannot be propagated until the export completes. Each deferred transaction is composed of one or more remote procedure calls (RPCs). If set to false and a transaction occurs that references objects in both unaffected master groups and master groups that are being extended, then the transaction may be split into two parts and sent to a destination in two separate transactions at different times. Such transactions are called split-transactions. If split-transactions are possible, then you must disable integrity constraints that may be violated by this behavior until the new master sites are added. If break_trans_to_masterdef is set to false, then existing masters cannot propagate their deferred transactions to the master definition site.
break_trans_to_new_masters	If break_trans_to_new_masters is set to true, then existing master sites may continue to propagate deferred transactions to the new master sites for replication groups that are not adding master sites. Each deferred transaction is composed of one or more remote procedure calls (RPCs). If set to true and a transaction occurs that references objects in both unaffected master groups and master groups that are being extended, then the transaction may be split into two parts and sent to a destination in two separate transactions at different times. Such transactions are called split-transactions. If split-transactions are possible, then you must disable integrity constraints that may be violated by this behavior until the new master sites are added. If break_trans_to_new_masters is set to false, then propagation of deferred transaction queues to the new masters is disabled.

Table 53–6 ADD_NEW_MASTERS Procedure Parameters

Parameter	Description
`percentage_for_catchup_mdef`	This parameter is meaningful only if `export_required` and `break_trans_to_masterdef` are both set to `true`. The percentage of propagation resources that should be used for catching up propagation to the master definition site. Must be a multiple of 10 and must be between 0 and 100.
`cycle_seconds_mdef`	This parameter is meaningful when `percentage_for_catchup_mdef` is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to the masterdef alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.
`percentage_for_catchup_new`	This parameter is meaningful only if `break_trans_to_new_masters` is set to `true`. The percentage of propagation resources that should be used for catching up propagation to new master sites. Must be a multiple of 10 and must be between 0 and 100.
`cycle_seconds_new`	This parameter is meaningful when `percentage_for_catchup_new` is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to a new master alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.

Exceptions

Table 53–7 ADD_NEW_MASTERS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
typefailure	The parameter value specified for one of the parameters is not appropriate.
novalidextreq	No valid extension request. The extension_id is not valid.
nonewsites	No new master sites to be added for the specified extension request.
notanewsite	Not a new site for extension request. A site was specified that was not specified when you ran the SPECIFY_NEW_MASTERS procedure.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.

Usage Notes

For a new master site to be instantiated using change-based recovery or full database export/import, the following conditions apply:

- The new master sites cannot have any existing replication groups.
- The master definition site cannot have any materialized view groups.
- The master definition site must be the same for all of the master groups. If one or more of these master groups have a different master definition site, then do not use change-based recovery or full database export/import. Use object-level export/import instead.
- The new master site must include all of the replication groups in the master definition site when the extension process is complete. That is, you cannot add a subset of the master groups at the master definition site to the new master site; all of the groups must be added.

> **Note:** To use change-based recovery, the existing master site and the new master site must be running under the same operating system, although the release of the operating system can differ.

For object-level export/import, before importing ensure that all the requests in the DBA_REPCATLOG data dictionary view for the extended groups have been processed without any error.

ADD_PRIORITY_*datatype* Procedure

This procedure adds a member to a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your priority column. You must call this procedure once for each of the possible values of the priority column.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.ADD_PRIORITY_datatype (
   gname              IN   VARCHAR2,
   pgroup             IN   VARCHAR2,
   value              IN   datatype,
   priority           IN   NUMBER);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 53–8 ADD_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Master group for which you are creating a priority group.
pgroup	Name of the priority group.
value	Value of the priority group member. This is one of the possible values of the associated priority column of a table using this priority group.
priority	Priority of this value. The higher the number, the higher the priority.

Exceptions

Table 53–9 ADD_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatevalue	Specified value already exists in the priority group.
duplicatepriority	Specified priority already exists in the priority group.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.
typefailure	Specified value has the incorrect datatype for the priority group.
notquiesced	Specified master group is not quiesced.

ADD_SITE_PRIORITY_SITE Procedure

This procedure adds a new site to a site priority group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
   gname          IN   VARCHAR2,
   name           IN   VARCHAR2
   site           IN   VARCHAR2,
   priority       IN   NUMBER);
```

Parameters

Table 53–10 ADD_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Master group for which you are adding a site to a group.
name	Name of the site priority group to which you are adding a member.
site	Global database name of the site that you are adding.
priority	Priority level of the site that you are adding. A higher number indicates a higher priority level.

Exceptions

Table 53–11 ADD_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Specified site priority group does not exist.
duplicatepriority	Specified priority level already exists for another site in the group.
duplicatevalue	Specified site already exists in the site priority group.
notquiesced	Master group is not quiesced.

ADD_*conflicttype*_RESOLUTION Procedure

These procedures designate a method for resolving an update, delete, or uniqueness conflict. You must call these procedures from the master definition site. The procedure that you need to call is determined by the type of conflict that the routine resolves.

Table 53–12 ADD_conflicttype_RESOLUTION Procedures

Conflict Type	Procedure Name
update	ADD_UPDATE_RESOLUTION
uniqueness	ADD_UNIQUE_RESOLUTION
delete	ADD_DELETE_RESOLUTION

See Also: *Oracle9i Replication* for more information about designating methods to resolve update conflicts, selecting uniqueness conflict resolution methods, and assigning delete conflict resolution methods

Syntax

```
DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   column_group           IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   method                 IN   VARCHAR2,
   parameter_column_name  IN   VARCHAR2
                               | DBMS_REPCAT.VARCHAR2s
                               | DBMS_UTILITY.LNAME_ARRAY,
   priority_group         IN   VARCHAR2     := NULL,
   function_name          IN   VARCHAR2     := NULL,
   comment                IN   VARCHAR2     := NULL);

DBMS_REPCAT.ADD_DELETE_RESOLUTION (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2s,
   function_name          IN   VARCHAR2,
   comment                IN   VARCHAR2     := NULL
   method                 IN   VARCHAR2     := 'USER FUNCTION');

DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   constraint_name        IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   method                 IN   VARCHAR2,
   parameter_column_name IN   VARCHAR2
                               | DBMS_REPCAT.VARCHAR2s
                               | DBMS_UTILITY.LNAME_ARRAY,
   function_name          IN   VARCHAR2     := NULL,
   comment                IN   VARCHAR2     := NULL);
```

Parameters

Table 53–13 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema containing the table to be replicated.
oname	Name of the table to which you are adding a conflict resolution routine. The table can be the storage table of a nested table.
column_group	Name of the column group to which you are adding a conflict resolution routine. Column groups are required for update conflict resolution routines only.
constraint_name	Name of the unique constraint or unique index for which you are adding a conflict resolution routine. Use the name of the unique index if it differs from the name of the associated unique constraint. Constraint names are required for uniqueness conflict resolution routines only.
sequence_no	Order in which the designated conflict resolution methods should be applied.
method	Type of conflict resolution routine that you want to create. This can be the name of one of the standard routines provided with advanced replication, or, if you have written your own routine, you should choose user function, and provide the name of your method as the function_name parameter. The standard methods supported in this release for update conflicts are: ▪ minimum ▪ maximum ▪ latest timestamp ▪ earliest timestamp ▪ additive, average ▪ priority group ▪ site priority ▪ overwrite ▪ discard The standard methods supported in this release for uniqueness conflicts are: append site name, append sequence, and discard. There are no built-in (Oracle supplied) methods for delete conflicts.

Table 53–13 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
parameter_column_name	Name of the columns used to resolve the conflict. The standard methods operate on a single column. For example, if you are using the latest timestamp method for a column group, then you should pass the name of the column containing the timestamp value as this parameter. If your are using a user function, then you can resolve the conflict using any number of columns. For update or unique conflicts, this parameter accepts either a comma-delimited list of column names, or a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMS_UTILITY.LNAME_ARRAY. Use DBMS_UTILITY.LNAME_ARRAY if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects. For delete conflicts, this parameter accepts either a comma-delimited list of column names or a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2. The single value '*' indicates that you want to use all of the columns in the table (or column group, for update conflicts) to resolve the conflict. If you specify '*', then the columns are passed to your function in alphabetical order. LOB columns cannot be specified for this parameter. **See Also:** "Usage Notes" on page 53-24 if you are using column objects
priority_group	If you are using the priority group or site priority update conflict resolution method, then you must supply the name of the priority group that you have created. See *Oracle9i Replication* for more information. If you are using a different method, you can use the default value for this parameter, NULL. This parameter is applicable to update conflicts only.
function_name	If you selected the user function method, or if you are adding a delete conflict resolution routine, then you must supply the name of the conflict resolution routine that you have written. If you are using one of the standard methods, then you can use the default value for this parameter, NULL.
comment	This user comment is added to the DBA_REPRESOLUTION view.

Exceptions

Table 53–14 ADD_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema using row-level replication.
missingschema	Specified schema does not exist.
missingcolumn	Column that you specified as part of the parameter_column_name parameter does not exist.
missinggroup	Specified column group does not exist.
missingprioritygroup	The priority group that you specified does not exist for the table.
invalidmethod	Resolution method that you specified is not recognized.
invalidparameter	Number of columns that you specified for the parameter_column_name parameter is invalid. (The standard routines take only one column name.)
missingfunction	User function that you specified does not exist.
missingconstraint	Constraint that you specified for a uniqueness conflict does not exist.
notquiesced	Replication group to which the specified table belongs is not quiesced.
duplicateresolution	Specified conflict resolution method is already registered.
duplicatesequence	The specified sequence number already exists for the specified object.
invalidprioritygroup	The specified priority group does not exist.
paramtype	Type is different from the type assigned to the priority group.

Usage Notes

If you are using column objects, then whether you can specify the attributes of the column objects for the `parameter_column_name` parameter depends on whether the conflict resolution method is built-in (Oracle supplied) or user-created:

- If you are using a built-in conflict resolution method, then you can specify attributes of objects for this parameter. For example, if a column object named `cust_address` has `street_address` as an attribute, then you can specify `cust_address.street_address` for this parameter.
- If you are using a built-in conflict resolution method, the following types of columns cannot be specified for this parameter: LOB attribute of a column object, collection or collection attribute of a column object, REF, or an entire column object.
- If you are using a user-created conflict resolution method, then you must specify an entire column object. You cannot specify the attributes of a column object. For example, if a column object named `cust_address` has `street_address` as an attribute (among other attributes), then you can specify only `cust_address` for this parameter.

ALTER_CATCHUP_PARAMETERS Procedure

This procedure alters the values for the following parameters stored in the DBA_REPEXTENSIONS data dictionary view:

- `percentage_for_catchup_mdef`
- `cycle_seconds_mdef`
- `percentage_for_catchup_new`
- `cycle_seconds_new`

These parameters were originally set by the ADD_NEW_MASTERS procedure. The new values you specify for these parameters are used during the remaining steps in the process of adding new master sites to a master group. These changes are only to the site at which it is executed. Therefore, it must be executed at each master site, including the master definition site, if you want to alter parameters at all sites.

See Also: "ADD_NEW_MASTERS Procedure" on page 53-10

Syntax

```
DBMS_REPCAT.ALTER_CATCHUP_PARAMETERS (
   extension_id                  IN    RAW,
   percentage_for_catchup_mdef   IN    BINARY_INTEGER  := NULL,
   cycle_seconds_mdef            IN    BINARY_INTEGER  := NULL,
   percentage_for_catchup_new    IN    BINARY_INTEGER  := NULL,
   cycle_seconds_new             IN    BINARY_INTEGER  := NULL);
```

Parameters

Table 53–15 ALTER_CATCHUP_PARAMETERS Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master database without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.
percentage_for_catchup_mdef	The percentage of propagation resources that should be used for catching up propagation to the master definition site. Must be a multiple of 10 and must be between 0 and 100.
cycle_seconds_mdef	This parameter is meaningful when percentage_for_catchup_mdef is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to the masterdef alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.
percentage_for_catchup_new	The percentage of propagation resources that should be used for catching up propagation to new master sites. Must be a multiple of 10 and must be between 0 and 100.
cycle_seconds_new	This parameter is meaningful when percentage_for_catchup_new is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to a new master alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.

Exceptions

Table 53–16 ALTER_CATCHUP_PARAMETERS Procedure Exceptions

Exception	Description
typefailure	The parameter value specified for one of the parameters is not appropriate.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.

ALTER_MASTER_PROPAGATION Procedure

This procedure alters the propagation method for a specified replication group at a specified master site. This replication group must be quiesced. You must call this procedure from the master definition site. If the master appears in the `dblink_list` or `dblink_table`, then `ALTER_MASTER_PROPAGATION` ignores that database link. You cannot change the propagation mode from a master to itself.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_PROPAGATION (
   gname               IN   VARCHAR2,
   master              IN   VARCHAR2,
   { dblink_list       IN   VARCHAR2,
   | dblink_table      IN   dbms_utility.dblink_array,}
   propagation_mode    IN   VARCHAR2 : ='asynchronous',
   comment             IN   VARCHAR2 := '');
```

Note: This procedure is overloaded. The `dblink_list` and `dblink_table` parameters are mutually exclusive.

Parameters

Table 53–17 ALTER_MASTER_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the replication group to which to alter the propagation mode.
master	Name of the master site at which to alter the propagation mode.
dblink_list	A comma-delimited list of database links for which to alter the propagation method. If NULL, then all masters except the master site being altered are used by default.
dblink_table	A PL/SQL index-by table, indexed from position 1, of database links for which to alter propagation.
propagation_mode	Determines the manner in which changes from the specified master site are propagated to the sites identified by the list of database links. Appropriate values are synchronous and asynchronous.
comment	This comment is added to the DBA_REPPROP view.

Exceptions

Table 53–18 ALTER_MASTER_PROPAGATION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Invocation site is not quiesced.
typefailure	Propagation mode specified was not recognized.
nonmaster	List of database links includes a site that is not a master site.

ALTER_MASTER_REPOBJECT Procedure

This procedure alters an object in your replication environment. You must call this procedure from the master definition site.

This procedure requires that you quiesce the master group of the object if either of the following conditions is true:

- You are altering a table in a multimaster replication environment.

- You are altering a table with the `safe_table_change` parameter set to `false` in a single master replication environment.

You can use this procedure to alter nontable objects without quiescing the master group.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   type                 IN   VARCHAR2,
   ddl_text             IN   VARCHAR2,
   comment              IN   VARCHAR2     := '',
   retry                IN   BOOLEAN      := false
   safe_table_change    IN   BOOLEAN      := false);
```

Parameters

Table 53–19 ALTER_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Schema containing the object that you want to alter.
oname	Name of the object that you want to alter. The object cannot be a storage table for a nested table.
type	Type of the object that you are altering. The following types are supported: FUNCTION SYNONYM INDEX TABLE INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
ddl_text	The DDL text that you want used to alter the object. Oracle does not parse this DDL before applying it. Therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being altered. If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.
comment	If not NULL, then this comment is added to the COMMENT field of the DBA_REPOBJECT view.
retry	If retry is true, then ALTER_MASTER_REPOBJECT alters the object only at masters whose object status is not VALID.

Table 53–19 ALTER_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
safe_table_change	Specify `true` if the change to a table is safe. Specify `false` if the change to a table is unsafe. You can make safe changes to a master table in a single master replication environment without quiescing the master group that contains the table. To make unsafe changes, you must quiesce the master group. Only specify this parameter for tables in single master replication environments. This parameter is ignored in multimaster replication environments and when the object specified is not a table. In multimaster replication environments, you must quiesce the master group to run the ALTER_MASTER_REPOBJECT procedure on a table. The following are safe changes: ■ Changing storage and extent information ■ Making existing columns larger. For example, changing a VARCHAR2(20) column to a VARCHAR2(50) column. ■ Adding non primary key constraints ■ Altering non primary key constraints ■ Enabling and disabling non primary key constraints The following are unsafe changes: ■ Changing the primary key by adding or deleting columns in the key ■ Adding or deleting columns ■ Making existing columns smaller. For example, changing a VARCHAR2(50) column to a VARCHAR2(20) column. ■ Disabling a primary key constraint ■ Changing the datatype of an existing column ■ Dropping an existing column If you are unsure whether a change is safe or unsafe, then quiesce the master group before you run the ALTER_MASTER_REPOBJECT procedure.

Exceptions

Table 53–20 ALTER_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Associated replication group has not been suspended.
missingobject	Object identified by sname and oname does not exist.
typefailure	Specified type parameter is not supported.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

ALTER_MVIEW_PROPAGATION Procedure

This procedure alters the propagation method for a specified replication group at the current materialized view site. This procedure pushes the deferred transaction queue at the materialized view site, locks the materialized view base tables, and regenerates any triggers and their associated packages. You must call this procedure from the materialized view site.

Syntax

```
DBMS_REPCAT.ALTER_MVIEW_PROPAGATION (
   gname                IN  VARCHAR2,
   propagation_mode     IN  VARCHAR2,
   comment              IN  VARCHAR2   := ''
   gowner               IN  VARCHAR2   := 'PUBLIC');
```

Parameters

Table 53–21 ALTER_MVIEW_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the replication group for which to alter the propagation method.
propagation_mode	Manner in which changes from the current materialized view site are propagated to its associated master site or master materialized view site. Appropriate values are synchronous and asynchronous.
comment	This comment is added to the DBA_REPPROP view.
gowner	Owner of the materialized view group.

Exceptions

Table 53–22 ALTER_MVIEW_PROPAGATION Procedure Exceptions

Exception	Description
missingrepgroup	Specified replication group does not exist.
typefailure	Propagation mode was specified incorrectly.
nonmview	Current site is not a materialized view site for the specified replication group.
commfailure	Cannot contact master site or master materialized view site.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
failaltermviewrop	Materialized view group propagation can be altered only when there are no other materialized view groups with the same master site or master materialized view site sharing the materialized view site.

ALTER_PRIORITY Procedure

This procedure alters the priority level associated with a specified priority group member. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY (
   gname            IN   VARCHAR2,
   pgroup           IN   VARCHAR2,
   old_priority     IN   NUMBER,
   new_priority     IN   NUMBER);
```

Parameters

Table 53–23 ALTER_PRIORITY Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group containing the priority that you want to alter.
old_priority	Current priority level of the priority group member.
new_priority	New priority level that you want assigned to the priority group member.

Exceptions

Table 53–24 ALTER_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatepriority	New priority level already exists in the priority group.
missingrepgroup	Specified master group does not exist.
missingvalue	Value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_*datatype*.
missingprioritygroup	Specified priority group does not exist.
notquiesced	Specified master group is not quiesced.

ALTER_PRIORITY_*datatype* Procedure

This procedure alters the value of a member in a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your priority column.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY_datatype (
   gname        IN   VARCHAR2,
   pgroup       IN   VARCHAR2,
   old_value    IN   datatype,
   new_value    IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 53–25 ALTER_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group containing the value that you want to alter.
old_value	Current value of the priority group member.
new_value	New value that you want assigned to the priority group member.

Exceptions

Table 53–26 ALTER_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatevalue	New value already exists in the priority group.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.
missingvalue	Old value does not exist.
paramtype	New value has the incorrect datatype for the priority group.
typefailure	Specified value has the incorrect datatype for the priority group.
notquiesced	Specified master group is not quiesced.

ALTER_SITE_PRIORITY Procedure

This procedure alters the priority level associated with a specified site. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods**:**

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY (
```

```
gname          IN   VARCHAR2,
name           IN   VARCHAR2,
old_priority   IN   NUMBER,
new_priority   IN   NUMBER);
```

Parameters

Table 53–27 ALTER_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Master group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_priority	Current priority level of the site whose priority level you want to change.
new_priority	New priority level for the site. A higher number indicates a higher priority level.

Exceptions

Table 53–28 ALTER_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Old priority level is not associated with any group members.
duplicatepriority	New priority level already exists for another site in the group.
missingvalue	Old value does not already exist.
paramtype	New value has the incorrect datatype for the priority group.
notquiesced	Master group is not quiesced.

ALTER_SITE_PRIORITY_SITE Procedure

This procedure alters the site associated with a specified priority level. You must call this procedure from the master definition site.

See Also: *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE (
   gname     IN   VARCHAR2,
   name      IN   VARCHAR2,
   old_site  IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 53–29 ALTER_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Master group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_site	Current global database name of the site to disassociate from the priority level.
new_site	New global database name that you want to associate with the current priority level.

Exceptions

Table 53–30 ALTER_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Specified site priority group does not exist.
missingvalue	Old site is not a group member.
notquiesced	Master group is not quiesced

CANCEL_STATISTICS Procedure

This procedure stops the collection of statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

Syntax

```
DBMS_REPCAT.CANCEL_STATISTICS (
   sname     IN   VARCHAR2,
   oname     IN   VARCHAR2);
```

Parameters

Table 53–31 CANCEL_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you do not want to gather conflict resolution statistics.

Exceptions

Table 53–32 CANCEL_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified table does not exist.
statnotreg	Specified table is not currently registered to collect statistics.

COMMENT_ON_COLUMN_GROUP Procedure

This procedure updates the comment field in the DBA_REPCOLUMN_GROUP view for a column group. This comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP (
   sname           IN   VARCHAR2,
   oname           IN   VARCHAR2,
   column_group    IN   VARCHAR2,
   comment         IN   VARCHAR2);
```

Parameters

Table 53–33 COMMENT_ON_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the replicated table with which the column group is associated.
column_group	Name of the column group.
comment	Text of the updated comment that you want included in the GROUP_COMMENT field of the DBA_REPCOLUMN_GROUP view.

Exceptions

Table 53–34 COMMENT_ON_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missinggroup	Specified column group does not exist.
missingobj	Object is missing.

COMMENT_ON_MVIEW_REPSITES Procedure

This procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP data dictionary view for the specified materialized view group. The group name must be registered locally as a replicated materialized view group. This procedure must be executed at the materialized view site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_MVIEW_REPSITES (
   gowner    IN   VARCHAR2,
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 53–35 COMMENT_ON_MVIEW_REPSITES Procedure Parameters

Parameter	Description
gowner	Owner of the materialized view group.
gname	Name of the materialized view group.
comment	Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Table 53–36 COMMENT_ON_MVIEW_REPSITES Procedure Exceptions

Parameter	Description
missingrepgroup	The materialized view group does not exist.
nonmview	The connected site is not a materialized view site.

COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Procedures

COMMENT_ON_PRIORITY_GROUP updates the comment field in the DBA_REPPRIORITY_GROUP view for a priority group. This comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

COMMENT_ON_SITE_PRIORITY updates the comment field in the DBA_REPPRIORITY_GROUP view for a site priority group. This procedure is a wrapper for the COMMENT_ON_COLUMN_GROUP procedure and is provided as a convenience only. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP (
   gname       IN   VARCHAR2,
   pgroup      IN   VARCHAR2,
   comment     IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2,
   comment     IN   VARCHAR2);
```

Parameters

Table 53–37 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Parameters

Parameter	Description
gname	Name of the master group.
pgroup/name	Name of the priority or site priority group.
comment	Text of the updated comment that you want included in the PRIORITY_COMMENT field of the DBA_REPPRIORITY_GROUP view.

Exceptions

Table 53–38 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.

COMMENT_ON_REPGROUP Procedure

This procedure updates the comment field in the DBA_REPGROUP view for a master group. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPGROUP (
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 53–39 COMMENT_ON_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group that you want to comment on.
comment	Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Exceptions

Table 53–40 COMMENT_ON_REPGROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPOBJECT Procedure

This procedure updates the comment field in the DBA_REPOBJECT view for a replicated object in a master group. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPOBJECT (
   sname    IN   VARCHAR2,
   oname    IN   VARCHAR2,
   type     IN   VARCHAR2,
   comment  IN   VARCHAR2);
```

Parameters

Table 53–41 COMMENT_ON_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to comment on. The object cannot be a storage table for a nested table.
type	Type of the object. The following types are supported: FUNCTION SYNONYM INDEX TABLE INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
comment	Text of the updated comment that you want to include in the OBJECT_COMMENT field of the DBA_REPOBJECT view.

Exceptions

Table 53–42 COMMENT_ON_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
typefailure	Specified type parameter is not supported.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPSITES Procedure

If the replication group is a master group, then this procedure updates the MASTER_COMMENT field in the DBA_REPSITES view for a master site. If the replication group is a materialized view group, this procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP view for a materialized view site.

This procedure can be executed at either a master site or a materialized view site. If you execute this procedure on a a materialized view site, then the materialized view group owner must be PUBLIC.

> **See Also:** "COMMENT_ON_conflicttype_RESOLUTION Procedure" on page 53-46 for instructions on placing a comment in the SCHEMA_COMMENT field of the DBA_REPGROUP view for a materialized view site if the materialized view group owner is not PUBLIC

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPSITES (
   gname       IN   VARCHAR2,
   [ master    IN   VARCHAR,]
   comment     IN   VARCHAR2);
```

Parameters

Table 53–43 COMMENT_ON_REPSITES Procedure Parameters

Parameter	Description
gname	Name of the replication group. This avoids confusion if a database is a master site in more than one replication environment.
master	The fully qualified database name of the master site on which you want to comment. If you are executing the procedure on a master site, then this parameter is required. To update comments at a materialized view site, omit this parameter. This parameter is optional.
comment	Text of the updated comment that you want to include in the comment field of the appropriate dictionary view. If the site is a master site, then this procedure updates the MASTER_COMMENT field of the DBA_REPSITES view. If the site is a materialized view site, then this procedure updates the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Exceptions

Table 53–44 COMMENT_ON_REPSITES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.

Table 53–44 COMMENT_ON_REPSITES Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible.
missingrepgroup	Replication group does not exist.
commfailure	One or more master sites are not accessible.
corrupt	There is an inconsistency in the replication catalog views.

COMMENT_ON_*conflicttype*_RESOLUTION Procedure

This procedure updates the RESOLUTION_COMMENT field in the DBA_REPRESOLUTION view for a conflict resolution routine. The procedure that you need to call is determined by the type of conflict that the routine resolves. These procedures must be issued at the master definition site.

Table 53–45 COMMENT_ON_conflicttype_RESOLUTION Procedures

Conflict Type	Procedure Name
update	COMMENT_ON_UPDATE_RESOLUTION
uniqueness	COMMENT_ON_UNIQUE_RESOLUTION
delete	COMMENT_ON_DELETE_RESOLUTION

The comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_UPDATE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   column_group         IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_UNIQUE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   constraint_name      IN   VARCHAR2,
```

```
   sequence_no           IN   NUMBER,
   comment               IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_DELETE_RESOLUTION (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   sequence_no           IN   NUMBER,
   comment               IN   VARCHAR2);
```

Parameters

Table 53–46 COMMENT_ON_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema.
oname	Name of the replicated table with which the conflict resolution routine is associated.
column_group	Name of the column group with which the update conflict resolution routine is associated.
constraint_name	Name of the unique constraint with which the uniqueness conflict resolution routine is associated.
sequence_no	Sequence number of the conflict resolution procedure.
comment	The text of the updated comment that you want included in the RESOLUTION_COMMENT field of the DBA_REPRESOLUTION view.

Exceptions

Table 53–47 COMMENT_ON_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
missingresolution	Specified conflict resolution routine is not registered.

COMPARE_OLD_VALUES Procedure

This procedure specifies whether to compare old column values during propagation of deferred transactions at each master site for each nonkey column of a replicated

table for updates and deletes. The default is to compare old values for all columns. You can change this behavior at all master sites and materialized view sites by invoking DBMS_REPCAT.COMPARE_OLD_VALUES at the master definition site.

When you use user-defined types, you can specify leaf attributes of a column object, or you can specify an entire column object. For example, if a column object named cust_address has street_address as an attribute, then you can specify cust_address.street_address for the column_list parameter or as part of the column_table parameter, or you can specify only cust_address.

When performing equality comparisons for conflict detection, Oracle treats objects as equal only if one of the following conditions is true:

- Both objects are atomically NULL (the entire object is NULL)
- All of the corresponding attributes are equal in the objects

Given these conditions, if one object is atomically NULL while the other is not, then Oracle does not consider the objects to be equal. Oracle does not consider MAP and ORDER methods when performing equality comparisons.

Syntax

```
DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   operation       IN  VARCHAR2 := 'UPDATE',
   compare         IN  BOOLEAN := true );
```

Note: This procedure is overloaded. The column_list and column_table parameters are mutually exclusive.

Parameters

Table 53–48 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table. The table can be the storage table of a nested table.
column_list	A comma-delimited list of the columns in the table. There must be no spaces between entries.
column_table	Instead of a list, you can use a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMS_UTILITY.LNAME_ARRAY to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use DBMS_UTILITY.LNAME_ARRAY if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.
operation	Possible values are: update, delete, or the asterisk wildcard '*', which means update and delete.
compare	If compare is true, the old values of the specified columns are compared when sent. If compare is false, the old values of the specified columns are not compared when sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is true for the table. The change takes effect at a master site or at a materialized view site the next time replication support is generated at that site with min_communication true.

Note: The operation parameter enables you to decide whether or not to compare old values for nonkey columns when rows are deleted or updated. If you do not compare the old value, then Oracle assumes the old value is equal to the current value of the column at the target side when the update or delete is applied.

See *Oracle9i Replication* for more information about reduced data propagation using the COMPARE_OLD_VALUES procedure before changing the default behavior of Oracle.

Exceptions

Table 53–49 COMPARE_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema waiting for row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Master group has not been quiesced.
typefailure	An illegal operation is specified.
keysendcomp	A specified column is a key column in a table.
dbnotcompatible	Feature is incompatible with database version. Typically, this exception arises when you are trying to compare the attributes of column objects. In this case, all databases must be at 9.0.1 or higher compatibility level.

CREATE_MASTER_REPGROUP Procedure

This procedure creates a new, empty, quiesced master group.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPGROUP (
   gname            IN   VARCHAR2,
   group_comment    IN   VARCHAR2     := '',
   master_comment   IN   VARCHAR2     := ''),
   qualifier        IN   VARCHAR2     := '');
```

Parameters

Table 53–50 CREATE_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the master group that you want to create.
group_comment	This comment is added to the DBA_REPGROUP view.
master_comment	This comment is added to the DBA_REPSITES view.
qualifier	Connection qualifier for master group. Be sure to use the @ sign. See *Oracle9i Replication* and *Oracle9i Database Administrator's Guide* for more information about connection qualifiers.

Exceptions

Table 53–51 CREATE_MASTER_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Master group already exists.
norepopt	Advanced replication option is not installed.
missingrepgroup	Master group name was not specified.
qualifiertoolong	Connection qualifier is too long.

CREATE_MASTER_REPOBJECT Procedure

This procedure makes an object a replicated object by adding the object to a master group. This procedure preserves the object identifier for user-defined types and object tables at all replication sites.

Replication of clustered tables is supported, but the use_existing_object parameter cannot be set to false for clustered tables. In other words, you must create the clustered table at all master sites participating in the master group before you execute the CREATE_MASTER_REPOBJECT procedure. However, these tables do not need to contain the table data. So, the copy_rows parameter can be set to true for clustered tables.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   type                 IN   VARCHAR2,
   use_existing_object  IN   BOOLEAN    := true,
   ddl_text             IN   VARCHAR2   := NULL,
   comment              IN   VARCHAR2   := '',
   retry                IN   BOOLEAN    := false
   copy_rows            IN   BOOLEAN    := true,
   gname                IN   VARCHAR2   := '');
```

Parameters

The following table describes the parameters for this procedure.

Table 53–52 CREATE_MASTER_REPOBJECT Procedure Parameters

Parameters	Description
sname	Name of the schema in which the object that you want to replicate is located.
oname	Name of the object you are replicating. If `ddl_text` is `NULL`, then this object must already exist in the specified schema. To ensure uniqueness, table names should be a maximum of 27 bytes long, and package names should be no more than 24 bytes. The object cannot be a storage table for a nested table.
type	Type of the object that you are replicating. The following types are supported: FUNCTION INDEX INDEXTYPE OPERATOR PACKAGE PACKAGE BODY PROCEDURE SYNONYM TABLE TRIGGER TYPE TYPE BODY VIEW

Table 53–52 CREATE_MASTER_REPOBJECT Procedure Parameters

Parameters	Description
use_existing_object	Indicate true if you want to reuse any objects of the same type and shape at the current master sites. See Table 53–54 for more information. **Note:** This parameter must be set to true for clustered tables.
ddl_text	If the object does not already exist at the master definition site, then you must supply the DDL text necessary to create this object. PL/SQL packages, package bodies, procedures, and functions must have a trailing semicolon. SQL statements do not end with trailing semicolon. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being created. If the DDL is supplied without specifying a schema (sname parameter), then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema. **Note:** Do not use the ddl_text parameter to add user-defined types or object tables. Instead, create the object first and then add the object.
comment	This comment is added to the OBJECT_COMMENT field of the DBA_REPOBJECT view.
retry	Indicate true if you want Oracle to reattempt to create an object that it was previously unable to create. Use this if the error was transient or has since been rectified, or if you previously had insufficient resources. If this is true, then Oracle creates the object only at master sites whose object status is not VALID.
copy_rows	Indicate true if you want the initial contents of a newly replicated object to match the contents of the object at the master definition site. See Table 53–54 for more information.
gname	Name of the replication group in which you want to create the replicated object. The schema name is used as the default replication group name if none is specified, and a replication group with the same name as the schema must exist for the procedure to complete successfully in that case.

Table 53–53 CREATE_MASTER_REPOBJECT Procedure Exceptions

Exceptions	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Master group is not quiesced.
duplicateobject	Specified object already exists in the master group and retry is false, or if a name conflict occurs.
missingobject	Object identified by sname and oname does not exist and appropriate DDL has not been provided.
typefailure	Objects of the specified type cannot be replicated.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.
notcompat	Not all remote masters in at least 7.3 compatibility mode.

Object Creations

Table 53–54 Object Creation at Master Sites

Object Already Exists?	COPY_ROWS	USE_EXISTING_OBJECTS	Result
yes	true	true	duplicatedobject message if objects do not match. For tables, use data from master definition site.
yes	false	true	duplicatedobject message if objects do not match. For tables, DBA must ensure contents are identical.
yes	true/false	false	duplicatedobject message.
no	true	true/false	Object is created. Tables populated using data from master definition site.
no	false	true/false	Object is created. DBA must populate tables and ensure consistency of tables at all sites.

CREATE_MVIEW_REPGROUP Procedure

This procedure creates a new, empty materialized view group in your local database. CREATE_MVIEW_REPGROUP automatically calls REGISTER_MIEW_REPGROUP, but ignores any errors that may have happened during registration.

Syntax

```
DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
   gname              IN   VARCHAR2,
   master             IN   VARCHAR2,
   comment            IN   VARCHAR2     := '',
   propagation_mode   IN   VARCHAR2     := 'ASYNCHRONOUS',
   fname              IN   VARCHAR2     := NULL
   gowner             IN   VARCHAR2     := 'PUBLIC');
```

Parameters

Table 53–55 CREATE_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group. This group must exist at the specified master site or master materialized view site.
master	Fully qualified database name of the database in the replication environment to use as the master site or master materialized view site. You can include a connection qualifier if necessary. See *Oracle9i Replication* and *Oracle9i Database Administrator's Guide* for information about using connection qualifiers.
comment	This comment is added to the DBA_REPGROUP view.
propagation_mode	Method of propagation for all updatable materialized views in the replication group. Acceptable values are synchronous and asynchronous.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
gowner	Owner of the materialized view group.

Exceptions

Table 53–56 CREATE_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Replication group already exists at the invocation site.
nonmaster	Specified database is not a master site or master materialized view site.
commfailure	Specified database is not accessible.
norepopt	Advanced replication option is not installed.
typefailure	Propagation mode was specified incorrectly.
missingrepgroup	Replication group does not exist at master site.
invalidqualifier	Connection qualifier specified for the master site or master materialized view site is not valid for the replication group.
alreadymastered	At the local site, there is another materialized view group with the same group name, but different master site or master materialized view site.

CREATE_MVIEW_REPOBJECT Procedure

This procedure adds a replicated object to a materialized view group.

Syntax

```
DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   type                   IN   VARCHAR2,
   ddl_text               IN   VARCHAR2  := '',
   comment                IN   VARCHAR2  := '',
   gname                  IN   VARCHAR2  := '',
   gen_objs_owner         IN   VARCHAR2  := '',
   min_communication      IN   BOOLEAN   := true,
   generate_80_compatible IN   BOOLEAN   := true,
   gowner                 IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–57 CREATE_MVIEW_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located. The schema must be same as the schema that owns the master table or master materialized view on which this materialized view is based.
oname	Name of the object that you want to add to the replicated materialized view group.
type	Type of the object that you are replicating. The following types are supported: FUNCTION SNAPSHOT INDEX SYNONYM INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
ddl_text	For objects of type SNAPSHOT, the DDL needed to create the object. For other types, use the default: '' (an empty string) If a materialized view with the same name already exists, then Oracle ignores the DDL and registers the existing materialized view as a replicated object. If the master table or master materialized view for a materialized view does not exist in the replication group of the master designated for this schema, then Oracle raises a missingobject error. If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema. If the object is not of type SNAPSHOT, then the materialized view site connects to the master site or master materialized view site and pulls down the DDL text to create the object. If the object type is TYPE or TYPE BODY, then the object identifier (OID) for the object at the materialized view site is the same as the OID at the master site or master materialized view site.

Table 53–57 CREATE_MVIEW_REPOBJECT Procedure Parameters

Parameter	Description
comment	This comment is added to the OBJECT_COMMENT field of the DBA_REPOBJECT view.
gname	Name of the replicated materialized view group to which you are adding an object. The schema name is used as the default group name if none is specified, and a materialized view group with the same name as the schema must exist for the procedure to complete successfully.
gen_objs_owner	Name of the user you want to assign as owner of the transaction.
min_communication	Set to false if the materialized view's master site is running Oracle7 release 7.3. Set to true to minimize new and old values of propagation. The default is true. For more information about conflict resolution methods, see *Oracle9i Replication.*
generate_80_compatible	Set to true if the materialized view's master site is running a version of Oracle server prior to Oracle8*i* release 8.1.5. Set to false if the materialized view's master site or master materialized view site is running Oracle8*i* release 8.1.5 or greater.
gowner	Owner of the materialized view group.

Exceptions

Table 53–58 CREATE_MVIEW_REPOBJECT Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Master is no longer a master site or master materialized view site.
missingobject	Specified object does not exist in the master's replication group.
duplicateobject	Specified object already exists with a different shape.
typefailure	Type is not an allowable type.
ddlfailure	DDL did not succeed.
commfailure	Master site or master materialized view site is not accessible.
missingschema	Schema does not exist as a database schema.
badmviewddl	DDL was executed but materialized view does not exist.
onlyonemview	Only one materialized view for master table or master materialized view can be created.
badmviewname	Materialized view base table differs from master table or master materialized view.
missingrepgroup	Replication group at the master does not exist.

DEFINE_COLUMN_GROUP Procedure

This procedure creates an empty column group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DEFINE_COLUMN_GROUP (
   sname            IN   VARCHAR2,
   oname            IN   VARCHAR2,
   column_group     IN   VARCHAR2,
   comment          IN   VARCHAR2 := NULL);
```

Parameters

Table 53–59 DEFINE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a column group.
column_group	Name of the column group that you want to create.
comment	This user text is displayed in the DBA_REPCOLUMN_GROUP view.

Exceptions

Table 53–60 DEFINE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
duplicategroup	Specified column group already exists for the table.
notquiesced	Replication group to which the specified table belongs is not quiesced.

DEFINE_PRIORITY_GROUP Procedure

This procedure creates a new priority group for a master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
   gname         IN   VARCHAR2,
   pgroup        IN   VARCHAR2,
   datatype      IN   VARCHAR2,
   fixed_length  IN   INTEGER := NULL,
   comment       IN   VARCHAR2 := NULL);
```

Parameters

Table 53–61 DEFINE_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Master group for which you are creating a priority group.
pgroup	Name of the priority group that you are creating.
datatype	Datatype of the priority group members. The datatypes supported are: CHAR, VARCHAR2, NUMBER, DATE, RAW, NCHAR, and NVARCHAR2.
fixed_length	You must provide a column length for the CHAR datatype. All other types can use the default, NULL.
comment	This user comment is added to the DBA_REPPRIORITY view.

Exceptions

Table 53–62 DEFINE_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
duplicatepriority group	Specified priority group already exists in the master group.
typefailure	Specified datatype is not supported.
notquiesced	Master group is not quiesced.

DEFINE_SITE_PRIORITY Procedure

This procedure creates a new site priority group for a master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY (
   gname        IN   VARCHAR2,
```

```
   name          IN   VARCHAR2,
   comment       IN   VARCHAR2 := NULL);
```

Parameters

Table 53–63 DEFINE_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	The master group for which you are creating a site priority group.
name	Name of the site priority group that you are creating.
comment	This user comment is added to the DBA_REPPRIORITY view.

Exceptions

Table 53–64 DEFINE_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
duplicate prioritygroup	Specified site priority group already exists in the master group.
notquiesced	Master group is not quiesced.

DO_DEFERRED_REPCAT_ADMIN Procedure

This procedure executes the local outstanding deferred administrative procedures for the specified master group at the current master site, or (with assistance from job queues) for all master sites.

DO_DEFERRED_REPCAT_ADMIN executes only those administrative requests submitted by the connected user who called DO_DEFERRED_REPCAT_ADMIN. Requests submitted by other users are ignored.

Syntax

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
   gname          IN   VARCHAR2,
   all_sites      IN   BOOLEAN := false);
```

Parameters

Table 53–65 DO_DEFERRED_REPCAT_ADMIN Procedure Parameters

Parameter	Description
gname	Name of the master group.
all_sites	If this is true, then use a job to execute the local administrative procedures at each master site.

Exceptions

Table 53–66 DO_DEFERRED_REPCAT_ADMIN Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible and all_sites is true.

DROP_COLUMN_GROUP Procedure

This procedure drops a column group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_COLUMN_GROUP (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   column_group IN   VARCHAR2);
```

Parameters

Table 53–67 DROP_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table whose column group you are dropping.
column_group	Name of the column group that you want to drop.

Exceptions

Table 53–68 DROP_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
referenced	Specified column group is being used in conflict detection and resolution.
missingobject	Specified table does not exist.
missinggroup	Specified column group does not exist.
notquiesced	Master group to which the table belongs is not quiesced.

DROP_GROUPED_COLUMN Procedure

This procedure removes members from a column group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2s);
```

Parameters

Table 53–69 DROP_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table in which the column group is located. The table can be the storage table of a nested table.
column_group	Name of the column group from which you are removing members.
list_of_column_names	Names of the columns that you are removing from the designated column group. This can either be a comma-delimited list or a PL/SQL index-by table of column names. The PL/SQL index-by table must be of type DBMS_REPCAT.VARCHAR2. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object, then you can specify SYS_NC_OID$ to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is a storage table of a nested table, then you can specify NESTED_TABLE_ID to add the column that tracks the identifier for each row of the nested table.

Exceptions

Table 53–70 DROP_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
notquiesced	Master group that the table belongs to is not quiesced.

DROP_MASTER_REPGROUP Procedure

This procedure drops a master group from your current site. To drop the master group from all master sites, including the master definition site, you can call this procedure at the master definition site, and set all_sites to true.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPGROUP (
   gname             IN VARCHAR2,
   drop_contents     IN BOOLEAN    := false,
   all_sites         IN BOOLEAN    := false);
```

Parameters

Table 53–71 DROP_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the master group that you want to drop from the current master site.
drop_contents	By default, when you drop the replication group at a master site, all of the objects remain in the database. They simply are no longer replicated. That is, the replicated objects in the replication group no longer send changes to, or receive changes from, other master sites. If you set this to true, then any replicated objects in the master group are dropped from their associated schemas.
all_sites	If this is true and if the invocation site is the master definition site, then the procedure synchronously multicasts the request to all masters. In this case, execution is immediate at the master definition site and may be deferred at all other master sites.

Exceptions

Table 53–72 DROP_MASTER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
nonmasterdef	Invocation site is not the master definition site and all_sites is true.
commfailure	At least one master site is not accessible and all_sites is true.
fullqueue	Deferred remote procedure call (RPC) queue has entries for the master group.
masternotremoved	Master does not recognize the master definition site and all_sites is true.

DROP_MASTER_REPOBJECT Procedure

This procedure drops a replicated object from a master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN     := false);
```

Parameters

Table 53–73 DROP_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to remove from the master group. The object cannot be a storage table for a nested table.
type	Type of object that you want to drop. The following types are supported: FUNCTION SYNONYM INDEX TABLE INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
drop_objects	By default, the object remains in the schema, but is dropped from the master group. That is, any changes to the object are no longer replicated to other master and materialized view sites. To completely remove the object from the replication environment, set this parameter to true. If the parameter is set to true, the object is dropped from the database at each master site.

Exceptions

Table 53–74 DROP_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
typefailure	Specified type parameter is not supported.
commfailure	At least one master site is not accessible.

DROP_MVIEW_REPGROUP Procedure

This procedure drops a materialized view site from your replication environment. DROP_MVIEW_REPGROUP automatically calls UNREGISTER_MVIEW_REPGROUP at the master site or master materialized view site to unregister the materialized view, but ignores any errors that may have occurred during unregistration. If DROP_MVIEW_REPGROUP is unsuccessful, then connect to the master site or master materialized view site and run UNREGISTER_MVIEW_REPGROUP.

Syntax

```
DBMS_REPCAT.DROP_MVIEW_REPGROUP (
   gname                IN   VARCHAR2,
   drop_contents        IN   BOOLEAN   := false
   gowner               IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–75 DROP_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group that you want to drop from the current materialized view site. All objects generated to support replication, such as triggers and packages, are dropped.
drop_contents	By default, when you drop the replication group at a materialized view site, all of the objects remain in their associated schemas. They simply are no longer replicated. If you set this to true, then any replicated objects in the replication group are dropped from their schemas.

Table 53–75 DROP_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gowner	Owner of the materialized view group.

Exceptions

Table 53–76 DROP_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
missingrepgroup	Specified replication group does not exist.

DROP_MVIEW_REPOBJECT Procedure

This procedure drops a replicated object from a materialized view site.

Syntax

```
DBMS_REPCAT.DROP_MVIEW_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN  := false);
```

Parameters

Table 53–77 DROP_MVIEW_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to drop from the replication group.
type	Type of the object that you want to drop. The following types are supported: FUNCTION SNAPSHOT INDEX SYNONYM INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
drop_objects	By default, the object remains in its associated schema, but is dropped from its associated replication group. To completely remove the object from its schema at the current materialized view site, set this parameter to true. If the parameter is set to true, the object is dropped from the database at the materialized view site.

Exceptions

Table 53–78 DROP_MVIEW_REPOBJECT Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
missingobject	Specified object does not exist.
typefailure	Specified type parameter is not supported.

DROP_PRIORITY Procedure

This procedure drops a member of a priority group by priority level. You must call this procedure from the master definition site.

See Also: *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_PRIORITY(
      gname          IN   VARCHAR2,
      pgroup         IN   VARCHAR2,
      priority_num   IN   NUMBER);
```

Parameters

Table 53–79 DROP_PRIORITY Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
priority_num	Priority level of the priority group member that you want to remove from the group.

Exceptions

Table 53–80 DROP_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.
notquiesced	Master group is not quiesced.

DROP_PRIORITY_GROUP Procedure

This procedure drops a priority group for a specified master group. You must call this procedure from the master definition site.

See Also: *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_GROUP (
   gname     IN   VARCHAR2,
   pgroup    IN   VARCHAR2);
```

Parameters

Table 53–81 DROP_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group that you want to drop.

Exceptions

Table 53–82 DROP_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
referenced	Specified priority group is being used in conflict resolution.
notquiesced	Specified master group is not quiesced.

DROP_PRIORITY_*datatype* Procedure

This procedure drops a member of a priority group by value. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your `priority` column.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_datatype (
   gname     IN   VARCHAR2,
   pgroup    IN   VARCHAR2,
   value     IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 53–83 DROP_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
value	Value of the priority group member that you want to remove from the group.

Exceptions

Table 53–84 DROP_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.
paramtype, typefailure	Value has the incorrect datatype for the priority group.
notquiesced	Specified master group is not quiesced

DROP_SITE_PRIORITY Procedure

This procedure drops a site priority group for a specified master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2);
```

Parameters

Table 53–85 DROP_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Master group with which the site priority group is associated.
name	Name of the site priority group that you want to drop.

Exceptions

Table 53–86 DROP_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
referenced	Specified site priority group is being used in conflict resolution.
notquiesced	Specified master group is not quiesced

DROP_SITE_PRIORITY_SITE Procedure

This procedure drops a specified site, by name, from a site priority group. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE (
   gname       IN   VARCHAR2,
```

```
   name        IN   VARCHAR2,
   site        IN   VARCHAR2);
```

Parameters

Table 53–87 DROP_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Master group with which the site priority group is associated.
name	Name of the site priority group whose member you are dropping.
site	Global database name of the site you are removing from the group.

Exceptions

Table 53–88 DROP_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Specified site priority group does not exist.
notquiesced	Specified master group is not quiesced.

DROP_*conflicttype*_RESOLUTION Procedure

This procedure drops an update, delete, or uniqueness conflict resolution routine. You must call these procedures from the master definition site. The procedure that you must call is determined by the type of conflict that the routine resolves.

Conflict Resolution Routines

The following table shows the procedure name for each conflict resolution routine.

Table 53–89 Conflict Resolution Routines

Routine	Procedure Name
update	DROP_UPDATE_RESOLUTION
uniqueness	DROP_UNIQUE_RESOLUTION
delete	DROP_DELETE_RESOLUTION

Syntax

```
DBMS_REPCAT.DROP_UPDATE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   column_group       IN   VARCHAR2,
   sequence_no        IN   NUMBER);

DBMS_REPCAT.DROP_DELETE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   sequence_no        IN   NUMBER);

DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   constraint_name    IN   VARCHAR2,
   sequence_no        IN   NUMBER);
```

Parameters

Table 53–90 DROP_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table for which you want to drop a conflict resolution routine.
column_group	Name of the column group for which you want to drop an update conflict resolution routine.
constraint_name	Name of the unique constraint for which you want to drop a unique conflict resolution routine.
sequence_no	Sequence number assigned to the conflict resolution method that you want to drop. This number uniquely identifies the routine.

Exceptions

Table 53–91 DROP_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema, or a conflict resolution routine with the specified sequence number is not registered.
notquiesced	Master group is not quiesced.

EXECUTE_DDL Procedure

This procedure supplies DDL that you want to have executed at some or all master sites. You can call this procedure only from the master definition site.

Syntax

```
DBMS_REPCAT.EXECUTE_DDL (
   gname           IN   VARCHAR2,
   { master_list   IN   VARCHAR2     := NULL,
    | master_table  IN   DBMS_UTILITY.DBLINK_ARRAY,}
   DDL_TEXT        IN   VARCHAR2);
```

Note: This procedure is overloaded. The `master_list` and `master_table` parameters are mutually exclusive.

Parameters

Table 53–92 EXECUTE_DDL Procedure Parameters

Parameter	Description
gname	Name of the master group.
master_list	A comma-delimited list of master sites at which you want to execute the supplied DDL. Do not put any spaces between site names. The default value, NULL, indicates that the DDL should be executed at all sites, including the master definition site.
master_table	A table that lists the master sites where you want to execute the supplied DDL. The first master should be at position 1, the second at position 2, and so on.
ddl_text	The DDL that you want to execute at each of the specified master sites. If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

Exceptions

Table 53–93 EXECUTE_DDL Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one site is not a master site.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

GENERATE_MVIEW_SUPPORT Procedure

This procedure activates triggers and generate packages needed to support the replication of updatable materialized views or procedural replication.You must call this procedure from the materialized view site.

Note: CREATE_MVIEW_REPOBJECT automatically generates materialized view support for updatable materialized views.

Syntax

```
DBMS_REPCAT.GENERATE_MVIEW_SUPPORT (
   sname                    IN VARCHAR2,
   oname                    IN VARCHAR2,
   type                     IN VARCHAR2,
   gen_objs_owner           IN VARCHAR2 := '',
   min_communication        IN BOOLEAN  := true,
   generate_80_compatible   IN BOOLEAN  := true);
```

Parameters

Table 53–94 GENERATE_MVIEW_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	The name of the object for which you are generating support.
type	Type of the object. The types supported are SNAPSHOT, PACKAGE, and PACKAGE BODY.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, the objects are created in SNAME.
min_communication	If true, then the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group.
generate_80_ compatible	Set to true if the materialized view's master site is running a version of Oracle server prior to Oracle8*i* release 8.1.5. Set to false if the materialized view's master site or master materialized view site is running Oracle8*i* release 8.1.5 or higher.

Exceptions

Table 53–95 GENERATE_MVIEW_SUPPORT Procedure Exceptions

Exceptions	Descriptions
nonmview	Invocation site is not a materialized view site.

Table 53–95 GENERATE_MVIEW_SUPPORT Procedure Exceptions

Exceptions	Descriptions
missingobject	Specified object does not exist as a materialized view in the replicated schema waiting for row / column-level replication information or as a package (body) waiting for wrapper generation.
typefailure	Specified type parameter is not supported.
missingschema	Specified owner of generated objects does not exist.
missingremoteobject	Object at master site or master materialized view site has not yet generated replication support.
commfailure	Master site or master materialized view site is not accessible.

GENERATE_REPLICATION_SUPPORT Procedure

This procedure generates the triggers and packages needed to support replication for a specified object. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
  sname                    IN    VARCHAR2,
  oname                    IN    VARCHAR2,
  type                     IN    VARCHAR2,
  package_prefix           IN    VARCHAR2   := NULL,
  procedure_prefix         IN    VARCHAR2   := NULL,
  distributed              IN    BOOLEAN    := true,
  gen_objs_owner           IN    VARCHAR2   := NULL,
  min_communication        IN    BOOLEAN    := true,
  generate_80_compatible   IN    BOOLEAN    := true);
```

Parameters

Table 53–96 GENERATE_REPLICATION_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	Name of the object for which you are generating replication support.
type	Type of the object. The types supported are: TABLE, PACKAGE, and PACKAGE BODY.
package_prefix	For objects of type PACKAGE or PACKAGE BODY this value is prepended to the generated wrapper package name. The default is DEFER_.
procedure_prefix	For objects of type PACKAGE or PACKAGE BODY, this value is prepended to the generated wrapper procedure names. By default, no prefix is assigned.
distributed	This must be set to true.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, the objects are created in sname.
min_communication	Set to false if any master site is running Oracle7 release 7.3. Set to true when you want propagation of new and old values to be minimized. The default is true. For more information, see *Oracle9i Replication*.
generate_80_compatible	Set to true if any master site is running a version of Oracle server prior to Oracle8*i* release 8.1.5. Set to false if all master sites are running Oracle8*i* release 8.1.5 or higher.

Exceptions

Table 53–97 GENERATE_REPLICATION_SUPPORT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema waiting for row-level replication information or as a package (body) waiting for wrapper generation.
typefailure	Specified type parameter is not supported.
notquiesced	Replication group has not been quiesced.
commfailure	At least one master site is not accessible.
missingschema	Schema does not exist.
dbnotcompatible	One of the master sites is not 7.3.0.0 compatible.
notcompat	One of the master sites is not 7.3.0.0 compatible. (Equivalent to dbnotcompatible.)
duplicateobject	Object already exists.

MAKE_COLUMN_GROUP Procedure

This procedure creates a new column group with one or more members. You must call this procedure from the master definition site.

> **See Also:** *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.MAKE_COLUMN_GROUP (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   column_group         IN   VARCHAR2,
   list_of_column_names IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2s);
```

Parameters

Table 53–98 MAKE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a new column group. The table can be the storage table of a nested table.
column_group	Name that you want assigned to the column group that you are creating.
list_of_column_names	Names of the columns that you are grouping. This can either be a comma-delimited list or a PL/SQL index-by table of column names. The PL/SQL index-by table must be of type DBMS_REPCAT.VARCHAR2. Use the single value '*' to create a column group that contains all of the columns in your table. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object table, then you can specify SYS_NC_OID$ to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is the storage table of a nested table, then you can specify NESTED_TABLE_ID to add the column that tracks the identifier for each row of the nested table.

Exceptions

Table 53–99 MAKE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicategroup	Specified column group already exists for the table.
missingobject	Specified table does not exist.
missingcolumn	Specified column does not exist in the designated table.
duplicatecolumn	Specified column is already a member of another column group.
notquiesced	Master group is not quiesced.

PREPARE_INSTANTIATED_MASTER Procedure

This procedure enables the propagation of deferred transactions from other prepared new master sites and existing master sites to the invocation master site. This procedure also enables the propagation of deferred transactions from the invocation master site to the other prepared new master sites and existing master sites.

If you performed a full database export/import or a change-based recovery, then the new master site includes all of the deferred transactions that were in the deferred transactions queue at the master definition site. Because these deferred transactions should not exist at the new master site, this procedure deletes all transactions in the deferred transactions queue and error queue if full database export/import or change-based recovery was used.

For object-level export/import, ensure that all the requests in the DBA_REPCATLOG data dictionary view for the extended groups have been processed without error before running this procedure.

Caution:

- Do not invoke this procedure until instantiation (export/import or change-based recovery) for the new master site is complete.
- Do not allow any data manipulation language (DML) statements directly on the objects in the extended master group in the new master site until execution of this procedure returns successfully. These DML statements may not be replicated.
- Do not use the DBMS_DEFER package to create deferred transactions until execution of this procedure returns successfully. These deferred transactions may not be replicated.

Note: To use change-based recovery, the existing master site and the new master site must be running under the same operating system, although the release of the operating system can differ.

Syntax

```
DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
   extension_id             IN       RAW);
```

Parameters

Table 53–100 PREPARE_INSTANTIATED_MASTER Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

Exceptions

Table 53–101 PREPARE_INSTANTIATED_MASTER Procedure Exceptions

Exception	Description
typefailure	The parameter value specified for one of the parameters is not appropriate.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.

PURGE_MASTER_LOG Procedure

This procedure removes local messages in the DBA_REPCATLOG view associated with a specified identification number, source, or master group.

To purge all of the administrative requests from a particular source, specify NULL for the id parameter. To purge all administrative requests from all sources, specify NULL for both the id parameter and the source parameter.

Syntax

```
DBMS_REPCAT.PURGE_MASTER_LOG (
   id      IN   BINARY_INTEGER,
   source  IN   VARCHAR2,
   gname   IN   VARCHAR2);
```

Parameters

Table 53–102 PURGE_MASTER_LOG Procedure Parameters

Parameter	Description
id	Identification number of the request, as it appears in the DBA_REPCATLOG view.
source	Master site from which the request originated.
gname	Name of the master group for which the request was made.

Exceptions

Table 53–103 PURGE_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	gname is not NULL, and the invocation site is not a master site.

PURGE_STATISTICS Procedure

This procedure removes information from the DBA_REPRESOLUTION_STATISTICS view.

Syntax

```
DBMS_REPCAT.PURGE_STATISTICS (
   sname      IN   VARCHAR2,
   oname      IN   VARCHAR2,
   start_date IN   DATE,
   end_date   IN   DATE);
```

Parameters

Table 53–104 PURGE_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the replicated table is located.
oname	Name of the table whose conflict resolution statistics you want to purge.
start_date/end_date	Range of dates for which you want to purge statistics. If start_date is NULL, then purge all statistics up to the end_date. If end_date is NULL, then purge all statistics after the start_date.

Exceptions

Table 53–105 PURGE_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified table does not exist.
statnotreg	Table not registered to collect statistics.

REFRESH_MVIEW_REPGROUP Procedure

This procedure refreshes a materialized view group with the most recent data from its associated master site or master materialized view site.

Syntax

```
DBMS_REPCAT.REFRESH_MVIEW_REPGROUP (
   gname                 IN   VARCHAR2,
   drop_missing_contents IN   BOOLEAN    := false,
   refresh_mviews        IN   BOOLEAN    := false,
   refresh_other_objects IN   BOOLEAN    := false,
   gowner                IN   VARCHAR2   := 'PUBLIC');
```

Parameters

Table 53–106 REFRESH_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group.
drop_missing_ contents	If an object was dropped from the replication group at the master site or master materialized view site, then it is not automatically dropped from the schema at the materialized view site. It is simply no longer replicated. That is, changes to this object are no longer sent to its associated master site or master materialized view site. Materialized views can continue to be refreshed from their associated master tables or master materialized views. However, any changes to an updatable materialized view are lost. When an object is dropped from the replication group, you can choose to have it dropped from the schema entirely by setting this parameter to true.
refresh_mviews	Set to true to refresh the contents of the materialized views in the replication group.
refresh_other_ objects	Set this to true to refresh the contents of the nonmaterialized view objects in the replication group. Nonmaterialized view objects may include the following: ▪ Tables ▪ Views ▪ Indexes ▪ PL/SQL packages and package bodies ▪ PL/SQL procedures and functions ▪ Triggers ▪ Synonyms
gowner	Owner of the materialized view group.

Exceptions

Table 53–107 REFRESH_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Master is no longer a master site or master materialized view site.
commfailure	Master site or master materialized view site is not accessible.

Table 53–107 REFRESH_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
missingrepgroup	Replication group name not specified.

REGISTER_MVIEW_REPGROUP Procedure

This procedure facilitates the administration of materialized views at their respective master sites or master materialized view sites by inserting or modifying a materialized view group in DBA_REGISTERED_MVIEW_GROUPS.

Syntax

```
DBMS_REPCAT.REGISTER_MVIEW_REPGROUP (
   gname            IN   VARCHAR2,
   mviewsite         IN   VARCHAR2,
   comment          IN   VARCHAR2  := NULL,
   rep_type         IN   NUMBER    := reg_unknown,
   fname            IN   VARCHAR2  := NULL
   gowner           IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–108 REGISTER_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the materialized view group to be registered.
mviewsite	Global name of the materialized view site.
comment	Comment for the materialized view site or update for an existing comment.
rep_type	Version of the materialized view group. Valid constants that can be assigned include the following: ▪ dbms_repcat.reg_unknown (the default) ▪ dbms_repcat.reg_v7_group ▪ dbms_repcat.reg_v8_group
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
gowner	Owner of the materialized view group.

Exceptions

Table 53–109 REGISTER_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
failregmviewrepgroup	Registration of materialized view group failed.
missingrepgroup	Replication group name not specified.
nullsitename	A materialized view site was not specified.
nonmaster	Procedure must be executed at the materialized view's master site or master materialized view site.
duplicaterepgroup	Replication group already exists.

REGISTER_STATISTICS Procedure

This procedure collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.

Syntax

```
DBMS_REPCAT.REGISTER_STATISTICS (
   sname IN   VARCHAR2,
   oname IN   VARCHAR2);
```

Parameters

Table 53–110 REGISTER_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you want to gather conflict resolution statistics.

Exceptions

Table 53–111 REGISTER_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified table does not exist.

RELOCATE_MASTERDEF Procedure

This procedure changes your master definition site to another master site in your replication environment.

It is not necessary for either the old or new master definition site to be available when you call RELOCATE_MASTERDEF. In a planned reconfiguration, you should invoke RELOCATE_MASTERDEF with notify_masters set to true and include_old_masterdef set to true.

Syntax

```
DBMS_REPCAT.RELOCATE_MASTERDEF (
   gname                 IN   VARCHAR2,
   old_masterdef         IN   VARCHAR2,
   new_masterdef         IN   VARCHAR2,
   notify_masters        IN   BOOLEAN    := true,
   include_old_masterdef IN   BOOLEAN    := true,
```

```
    require_flavor_change  IN   BOOLEAN    := false);
```

Parameters

Table 53–112 RELOCATE_MASTERDEF Procedure Parameters

Parameter	Description
gname	Name of the replication group whose master definition you want to relocate.
old_masterdef	Fully qualified database name of the current master definition site.
new_masterdef	Fully qualified database name of the existing master site that you want to make the new master definition site.
notify_masters	If this is true, then the procedure synchronously multicasts the change to all masters (including old_masterdef only if include_old_masterdef is true). If any master does not make the change, then roll back the changes at all masters. If just the master definition site fails, then you should invoke RELOCATE_MASTERDEF with notify_masters set to true and include_old_masterdef set to false. If several master sites and the master definition site fail, then the administrator should invoke RELOCATE_MASTERDEF at each operational master with notify_masters set to false.
include_old_masterdef	If notify_masters is true and if include_old_masterdef is also true, then the old master definition site is also notified of the change.
require_flavor_change	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 53–113 RELOCATE_MASTERDEF Procedure Exceptions

Exception	Description
nonmaster	new_masterdef is not a master site or the invocation site is not a master site.
nonmasterdef	old_masterdef is not the master definition site.
commfailure	At least one master site is not accessible and notify_masters is true.

REMOVE_MASTER_DATABASES Procedure

This procedure removes one or more master databases from a replication environment. This procedure regenerates the triggers and their associated packages at the remaining master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.REMOVE_MASTER_DATABASES (
   gname           IN   VARCHAR2,
   master_list     IN   VARCHAR2 |
   master_table    IN   DBMS_UTILITY.DBLINK_ARRAY);
```

> **Note:** This procedure is overloaded. The `master_list` and `master_table` parameters are mutually exclusive.

Parameters

Table 53–114 REMOVE_MASTER_DATABASES Procedure Parameters

Parameter	Description
`gname`	Name of the replication group associated with the replication environment. This prevents confusion if a master database is involved in more than one replication environment.
`master_list`	A comma-delimited list of fully qualified master database names that you want to remove from the replication environment. There must be no spaces between names in the list.
`master_table`	In place of a list, you can specify the database names in a PL/SQL index-by table of type `DBMS_UTILITY.DBLINK_ARRAY`.

Exceptions

Table 53–115 REMOVE_MASTER_DATABASES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one of the specified databases is not a master site.
reconfigerror	One of the specified databases is the master definition site.
commfailure	At least one remaining master site is not accessible.

RENAME_SHADOW_COLUMN_GROUP Procedure

This procedure renames the shadow column group of a replicated table to make it a named column group. The replicated table's master group does not need to be quiesced to run this procedure.

Syntax

```
DBMS_REPCAT.RENAME_SHADOW_COLUMN_GROUP (
   sname                IN VARCHAR2,
   oname                IN VARCHAR2,
   new_col_group_name   IN VARCHAR2)
```

Parameters

Table 53–116 RENAME_SHADOW_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table.
new_col_group_ name	Name of the new column group. The columns currently in the shadow group are placed in a column group with the name you specify.

Exceptions

Table 53–117 RENAME_SHADOW_COLUMN_GROUP Procedure Exceptions

Exception	Description
missmview	The specified schema does not exist.
nonmasterdef	Invocation site is not the master definition site.
missingobject	The specified object does not exist.
duplicategroup	The column group that was specified for creation already exists.

REPCAT_IMPORT_CHECK Procedure

This procedure ensures that the objects in the master group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by Oracle Replication.

Syntax

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK (
   gname      IN   VARCHAR2,
   master     IN   BOOLEAN,
   gowner     IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–118 REPCAT_IMPORT_CHECK Procedure Parameters

Parameter	Description
gname	Name of the master group. If you omit both parameters, then the procedure checks all master groups at your current site.
master	Set this to `true` if you are checking a master site and `false` if you are checking a materialized view site.
gowner	Owner of the master group.

Exceptions

Table 53–119 REPCAT_IMPORT_CHECK Procedure Exceptions

Exception	Description
nonmaster	master is true and either the database is not a master site for the replication group or the database is not the expected database.
nonmview	master is false and the database is not a materialized view site for the replication group.
missingobject	A valid replicated object in the replication group does not exist.
missingrepgroup	The specified replicated replication group does not exist.
missingschema	The specified replicated replication group does not exist.

RESUME_MASTER_ACTIVITY Procedure

This procedure resumes normal replication activity after quiescing a replication environment.

Syntax

```
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
   gname       IN  VARCHAR2,
   override    IN  BOOLEAN := false);
```

Parameters

Table 53–120 RESUME_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the master group.
override	If this is true, then it ignores any pending RepCat administrative requests and restores normal replication activity at each master as quickly as possible. This should be considered only in emergency situations. If this is false, then it restores normal replication activity at each master only when there is no pending RepCat administrative request for gname at that master.

Exceptions

Table 53–121 RESUME_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Master group is not quiescing or quiesced.
commfailure	At least one master site is not accessible.
notallgenerated	Generate replication support before resuming replication activity.

RESUME_PROPAGATION_TO_MDEF Procedure

During the process of adding new master sites to a master group without quiesce, this procedure indicates that export is effectively finished and propagation to the master definition site for both extended and unaffected replication groups existing at master sites can be enabled. Run this procedure after the export required to add new master sites to a master group is complete.

Syntax

```
DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
   extension_id                IN    RAW);
```

Parameters

Table 53–122 RESUME_PROPAGATION_TO_MDEF Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

Exceptions

Table 53–123 RESUME_PROPAGATION_TO_MDEF Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
extstinapp	Extension status is inappropriate. The extension status should be EXPORTING when you run this procedure. To check the extension status, query the DBA_REPEXTENSIONS data dictionary view.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.

SEND_OLD_VALUES Procedure

You have the option of sending old column values during propagation of deferred transactions for each nonkey column of a replicated table when rows are updated or deleted in the table. When min_communication is set to true, the default is the following:

- For a deleted row, to send old values for all columns
- For an updated row, to send old values for key columns and the modified columns in a column group

You can change this behavior at all master sites and materialized view sites by invoking DBMS_REPCAT.SEND_OLD_VALUES at the master definition site. Then, generate replication support at all master sites and at each materialized view site.

When you use user-defined types, you can specify the leaf attributes of a column object, or an entire column object. For example, if a column object named cust_address has street_address as an attribute, then you can specify cust_address.street_address for the column_list parameter or as part of the column_table parameter, or you can specify only cust_address.

Syntax

```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   operation       IN  VARCHAR2 := 'UPDATE',
   send            IN  BOOLEAN  := true );
```

Note: This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Parameters

Table 53–124 SEND_OLD_VALUES Procedure Parameters

Parameter	Description
`sname`	Schema in which the table is located.
`oname`	Name of the replicated table. The table can be the storage table of a nested table.
`column_list`	A comma-delimited list of the columns in the table. There must be no spaces between entries.
`column_table`	Instead of a list, you can use a PL/SQL index-by table of type `DBMS_REPCAT.VARCHAR2` or `DBMS_UTILITY.LNAME_ARRAY` to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use `DBMS_UTILITY.LNAME_ARRAY` if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.
`operation`	Possible values are: `update`, `delete`, or the asterisk wildcard `'*'`, which means update and delete.
`send`	If `true`, then the old values of the specified columns are sent. If `false`, then the old values of the specified columns are not sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as `min_communication` is `true` for the table. The change takes effect at a master site or at a materialized view site the next time replication support is generated at that site with `min_communication true`.

Note: The `operation` parameter enables you to specify whether or not to transmit old values for nonkey columns when rows are deleted or updated. If you do not send the old value, then Oracle sends a `NULL` in place of the old value and assumes the old value is equal to the current value of the column at the target side when the update or delete is applied.

See *Oracle9i Replication* for information about reduced data propagation using the `SEND_OLD_VALUES` procedure before changing the default behavior of Oracle.

Exceptions

Table 53–125 SEND_OLD_VALUES Procedure Exceptions

Exception	Description
`nonmasterdef`	Invocation site is not the master definition site.
`missingobject`	Specified object does not exist as a table in the specified schema waiting for row-level replication information.
`missingcolumn`	At least one column is not in the table.
`notquiesced`	Master group has not been quiesced.
`typefailure`	An illegal operation is specified.
`keysendcomp`	A specified column is a key column in a table.
`dbnotcompatible`	Feature is incompatible with database version. Typically, this exception arises when you are trying to send the attributes of column objects. In this case, all databases must be at 9.0.1 or higher compatibility level.

SET_COLUMNS Procedure

This procedure enables you to use an alternate column or group of columns, instead of the primary key, to determine which columns of a table to compare when using row-level replication. You must call this procedure from the master definition site.

When you use column objects, if an attribute of a column object can be used as a primary key or part of a primary key, then the attribute can be part of an alternate key column. For example, if a column object named `cust_address` has `street_address` as a `VARCHAR2` attribute, then you can specify `cust_address.street_`

address for the column_list parameter or as part of the column_table parameter. However, the entire column object, cust_address, cannot be specified.

For the storage table of a nested table column, this procedure accepts the NESTED_TABLE_ID as an alternate key column.

When you use object tables, you cannot specify alternate key columns. If the object identifier (OID) is system-generated for an object table, then Oracle uses the OID column in the object table as the key for the object table. If the OID is user-defined for an object table, then Oracle uses the primary key in the object table as the key.

The following types of columns cannot be alternate key columns:

- LOB or LOB attribute of a column object
- Collection or collection attribute of a column object
- REF
- An entire column object

> **See Also:** The *constraint_clause* in *Oracle9i SQL Reference* for more information about restrictions on primary key columns

Syntax

```
DBMS_REPCAT.SET_COLUMNS (
   sname           IN    VARCHAR2,
   oname           IN    VARCHAR2,
   { column_list   IN    VARCHAR2
   | column_table  IN    DBMS_UTILITY.NAME_ARRAY | DBMS_UTILITY.LNAME_ARRAY } );
```

> **Note:** This procedure is overloaded. The column_list and column_table parameters are mutually exclusive.

Parameters

Table 53–126 SET_COLUMNS Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table.
column_list	A comma-delimited list of the columns in the table that you want to use as a primary key. There must be no spaces between entries.
column_table	Instead of a list, you can use a PL/SQL index-by table of type DBMS_UTILITY.NAME_ARRAY or DBMS_UTILITY.LNAME_ARRAY to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use DBMS_UTILITY.LNAME_ARRAY if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.

Exceptions

Table 53–127 SET_COLUMNS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema waiting for row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replication group is not quiescing or quiesced.

SPECIFY_NEW_MASTERS Procedure

This procedure specifies the master sites you intend to add to an existing replication group without quiescing the group. This procedure must be run at the master definition site of the specified master group.

If necessary, this procedure creates an extension_id that tracks the process of adding new master sites to a master group. You use this extension_id in the other procedures that you run at various stages in the process. You can view

information about the extension_id in the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

This procedure adds the new master sites to the DBA_REPSITES_NEW data dictionary view for the specified replication group. This procedure can be run any number of times for a given replication group. If it is run more than once, then it replaces any masters in the local DBA_REPSITES_NEW data dictionary view for the specified replication group with the masters specified in the master_list/master_table parameters.

You must run this procedure before you run the ADD_NEW_MASTERS procedure. No new master sites are added to the master group until you run the ADD_NEW_MASTERS procedure.

See Also: "ADD_NEW_MASTERS Procedure" on page 53-10

Syntax

```
DBMS_REPCAT.SPECIFY_NEW_MASTERS (
   gname           IN    VARCHAR2,
   { master_list   IN    VARCHAR2
    | master_table  IN   DBMS_UTILITY.DBLINK_ARRAY});
```

Note: This procedure is overloaded. The master_list and master_table parameters are mutually exclusive.

Parameters

Table 53–128 SPECIFY_NEW_MASTERS Procedure Parameters

Parameter	Description
gname	Master group to which you are adding new master sites.
master_list	A comma-delimited list of new master sites that you want to add to the master group. List only the new master sites, not the existing master sites. Do not put any spaces between site names. If master_list is NULL, all master sites for the given replication group are removed from the DBA_REPSITES_NEW data dictionary view. Specify NULL to indicate that the master group is not being extended.
master_table	A table that lists the new master sites that you want to add to the master group. In the table, list only the new master sites, not the existing master sites. The first master site should be at position 1, the second at position 2, and so on. If the table is empty, then all master sites for the specified replication group are removed from the DBA_REPSITES_NEW data dictionary view. Use an empty table to indicate that the master group is not being extended.

Exceptions

Table 53–129 SPECIFY_NEW_MASTERS Procedure Exceptions

Exception	Description
duplicaterepgroup	A master site that you are attempting to add is already part of the master group.
nonmasterdef	Invocation site is not the master definition site.
propmodenotallowed	Synchronous propagation mode not allowed for this operation. Only asynchronous propagation mode is allowed.
extstinapp	Extension request with status not allowed. There must either be no extension_id for the master group or the extension_id status must be READY. You can view the status for each extension_id at a master site in the DBA_REPEXTENSIONS data dictionary view.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.
notsamecq	Master groups do not have the same connection qualifier.

SUSPEND_MASTER_ACTIVITY Procedure

This procedure suspends replication activity for a master group. You use this procedure to quiesce the master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
   gname    IN    VARCHAR2);
```

Parameters

Table 53–130 SUSPEND_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the master group for which you want to suspend activity.

Exceptions

Table 53–131 SUSPEND_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notnormal	Master group is not in normal operation.
commfailure	At least one master site is not accessible.

SWITCH_MVIEW_MASTER Procedure

This procedure changes the master site of a materialized view group to another master site. This procedure does a full refresh of the affected materialized views and regenerates the triggers and their associated packages as needed. This procedure does not push the queue to the old master site before changing master sites.

If `min_communication` is `true` for the materialized view and the new master site is an Oracle7 master site, then regenerate replication support for the materialized view with `min_communication` set to `false`.

If `generate_80_compatible` is `false` for the materialized view and the new master site is a release lower than Oracle8*i* (Oracle7 or Oracle8), then regenerate replication support for the materialized view with `generate_80_compatible` set to `true`.

You can set both parameters for a materialized view in one call to `DBMS_REPCAT.GENERATE_MVIEW_SUPPORT`.

Note: You cannot switch the master of materialized views that are based on other materialized views (level 2 and greater materialized views). Such a materialized view must be dropped and re-created if you want to base it on a different master.

See Also: "GENERATE_MVIEW_SUPPORT Procedure" on page 53-78

Syntax

```
DBMS_REPCAT.SWITCH_MVIEW_MASTER (
   gname          IN   VARCHAR2,
   master         IN   VARCHAR2,
   gowner         IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–132 SWITCH_MVIEW_MASTER Procedure Parameters

Parameter	Description
`gname`	Name of the materialized view group for which you want to change the master site.
`master`	Fully qualified database name of the new master site to use for the materialized view group.
`gowner`	Owner of the materialized view group.

Exceptions

Table 53–133 SWITCH_MVIEW_MASTER Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Specified database is not a master site.
commfailure	Specified database is not accessible.
missingrepgroup	Materialized view group does not exist.
qrytoolong	Materialized view definition query is greater 32 KB.
alreadymastered	At the local site, there is another materialized view group with the same group name mastered at the old master site.

UNDO_ADD_NEW_MASTERS_REQUEST Procedure

This procedure undoes all of the changes made by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures for a specified extension_id.

This procedure is executed at one master site, which may be the master definition site, and it only affects that master site. If you run this procedure at one master site affected by the request, you must run it at all new and existing master sites affected by the request. You can query the DBA_REPSITES_NEW data dictionary view to see the new master sites affected by the extension_id. This data dictionary view also lists the replication group name, and you must run this procedure at all existing master sites in the replication group.

Caution: This procedure is not normally called. Use this procedure only if the adding new masters without quiesce operation cannot proceed at one or more master sites. Run this procedure after you have already run the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures, but *before* you have run the RESUME_PROPAGATION_TO_MDEF and PREPARE_INSTANTIATED_MASTER procedures.

Do not run this procedure after you have run either RESUME_PROPAGATION_TO_MDEF or PREPARE_INSTANTIATED_MASTER for a particular extension_id.

See Also:

- "SPECIFY_NEW_MASTERS Procedure" on page 53-102
- "ADD_NEW_MASTERS Procedure" on page 53-10
- "RESUME_PROPAGATION_TO_MDEF Procedure" on page 53-97
- "PREPARE_INSTANTIATED_MASTER Procedure" on page 53-84

Syntax

```
DBMS_REPCAT.UNDO_ADD_NEW_MASTERS_REQUEST (
   extension_id         IN RAW,
   drop_contents        IN BOOLEAN := TRUE);
```

Parameters

Table 53–134 UNDO_ADD_NEW_MASTERS_REQUEST Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.
drop_contents	Specify true, the default, to drop the contents of objects in new replication groups being extended at the local site. Specify false to retain the contents.

Exceptions

Table 53–135 UNDO_ADD_NEW_MASTERS_REQUEST Procedure Exceptions

Exception	Description
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.1 or higher compatibility level.
typefail	A parameter value that you specified is not appropriate.

UNREGISTER_MVIEW_REPGROUP Procedure

This procedure facilitates the administration of materialized views at their respective master sites or master materialized view sites by deleting a materialized view group from DBA_REGISTERED_MVIEW_GROUPS. Run this procedure at the master site or master materialized view site.

Syntax

```
DBMS_REPCAT.UNREGISTER_MVIEW_REPGROUP (
   gname      IN   VARCHAR2,
   mviewsite  IN   VARCHAR2
   gowner     IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 53–136 UNREGISTER_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the materialized view group to be unregistered.
mviewsite	Global name of the materialized view site.
gowner	Owner of the materialized view group.

VALIDATE Function

This function validates the correctness of key conditions of a multimaster replication environment.

Syntax

```
DBMS_REPCAT.VALIDATE (
   gname               IN  VARCHAR2,
   check_genflags      IN  BOOLEAN := false,
   check_valid_objs    IN  BOOLEAN := false,
   check_links_sched   IN  BOOLEAN := false,
   check_links         IN  BOOLEAN := false,
   error_table         OUT DBMS_REPCAT.VALIDATE_ERR_TABLE)
  RETURN BINARY_INTEGER;
```

```
DBMS_REPCAT.VALIDATE (
   gname               IN  VARCHAR2,
   check_genflags      IN  BOOLEAN := false,
   check_valid_objs    IN  BOOLEAN := false,
   check_links_sched   IN  BOOLEAN := false,
   check_links         IN  BOOLEAN := false,
   error_msg_table     OUT DBMS_UTILITY.UNCL_ARRAY,
   error_num_table     OUT DBMS_UTILITY.NUMBER_ARRAY )
  RETURN BINARY_INTEGER;
```

Note: This function is overloaded. The return value of VALIDATE is the number of errors found. The function's OUT parameter returns any errors that are found. In the first interface function shown under "Syntax" on page 53-109, the error_table consists of an array of records. Each record has a VARCHAR2 and a NUMBER in it. The string field contains the error message, and the number field contains the Oracle error number.

The second interface function shown under "Syntax" on page 53-109 is similar except that there are two OUT arrays: a VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

Parameters

Table 53–137 VALIDATE Function Parameters

Parameter	Description
gname	Name of the master group to validate.
check_genflags	Check whether all the objects in the group are generated. This must be done at the master definition site only.
check_valid_objs	Check that the underlying objects for objects in the group valid. This must be done at the master definition site only. The master definition site goes to all other sites and checks that the underlying objects are valid. The validity of the objects is checked within the schema of the connected user.
check_links_sched	Check whether the links are scheduled for execution. This should be invoked at each master site.

Table 53–137 VALIDATE Function Parameters

Parameter	Description
check_links	Check whether the connected user (repadmin), as well as the propagator, have correct links for replication to work properly. Checks that the links exist in the database and are accessible. This should be invoked at each master site.
error_table	Returns the messages and numbers of all errors that are found.
error_msg_table	Returns the messages of all errors that are found.
error_num_table	Returns the numbers of all errors that are found.

Exceptions

Table 53–138 VALIDATE Function Exceptions

Exception	Description
missingdblink	Database link does not exist in the schema of the replication propagator or has not been scheduled. Ensure that the database link exists in the database, is accessible, and is scheduled for execution.
dblinkmismatch	Database link name at the local node does not match the global name of the database that the link accesses. Ensure that the GLOBAL_NAMES initialization parameter is set to true and the link name matches the global name.
dblinkuidmismatch	User name of the replication administration user at the local node and the user name at the node corresponding to the database link are not the same. Oracle Replication expects the two users to be the same. Ensure that the user identification of the replication administration user at the local node and the user identification at the node corresponding to the database link are the same.
objectnotgenerated	Object has not been generated at other master sites or is still being generated. Ensure that the object is generated by calling GENERATE_REPLICATION_SUPPORT and DO_DEFERRED_REPCAT_ADMIN for the object at the master definition site.
opnotsupported	Operation is not supported if the replication group is replicated at a pre-Oracle8 node. Ensure that all nodes of the master group are running Oracle8 and higher.

Usage Notes

The return value of VALIDATE is the number of errors found. The function's OUT parameter returns any errors that are found. In the first interface function, the error_table consists of an array of records. Each record has a VARCHAR2 and a NUMBER in it. The string field contains the error message and the number field contains the Oracle error number.

The second interface is similar except that there are two OUT arrays. A VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

WAIT_MASTER_LOG Procedure

This procedure determines whether changes that were asynchronously propagated to a master site have been applied.

Syntax

```
DBMS_REPCAT.WAIT_MASTER_LOG (
   gname          IN    VARCHAR2,
   record_count   IN    NATURAL,
   timeout        IN    NATURAL,
   true_count     OUT   NATURAL);
```

Parameters

Table 53–139 WAIT_MASTER_LOG Procedure Parameters

Parameter	Description
gname	Name of the master group.
record_count	Procedure returns whenever the number of incomplete activities is at or below this threshold.
timeout	Maximum number of seconds to wait before the procedure returns.
true_count (out parameter)	Returns the number of incomplete activities.

Exceptions

Table 53–140 WAIT_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.

54

DBMS_REPCAT_ADMIN

DBMS_REPCAT_ADMIN enables you to create users with the privileges needed by the symmetric replication facility.

This chapter discusses the following topics:

Summary of DBMS_REPCAT_ADMIN Subprograms

Table 54–1 DBMS_REPCAT_ADMIN Package Subprograms

Subprogram	Description
GRANT_ADMIN_ANY_SCHEMA Procedure on page 54-2	Grants the necessary privileges to the replication administrator to administer any replication group at the current site.
GRANT_ADMIN_SCHEMA Procedure on page 54-3	Grants the necessary privileges to the replication administrator to administer a schema at the current site.
REGISTER_USER_REPGROUP Procedure on page 4	Assigns proxy materialized view administrator or receiver privileges at the master site or master materialized view site for use with remote sites.
REVOKE_ADMIN_ANY_SCHEMA Procedure on page 54-6	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_ANY_SCHEMA.
REVOKE_ADMIN_SCHEMA Procedure on page 54-6	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_SCHEMA.
UNREGISTER_USER_REPGROUP Procedure on page 54-7	Revokes the privileges and roles from the proxy materialized view administrator or receiver that were granted by the REGISTER_USER_REPGROUP procedure.

GRANT_ADMIN_ANY_SCHEMA Procedure

This procedure grants the necessary privileges to the replication administrator to administer any replication groups at the current site.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
   username  IN   VARCHAR2);
```

Parameters

Table 54–2 GRANT_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator to whom you want to grant the necessary privileges and roles to administer any replication groups at the current site.

Exceptions

Table 54–3 GRANT_ADMIN_ANY_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

GRANT_ADMIN_SCHEMA Procedure

This procedure grants the necessary privileges to the replication administrator to administer a schema at the current site. This procedure is most useful if your replication group does not span schemas.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 54–4 GRANT_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator. This user is then granted the necessary privileges and roles to administer the schema of the same name within a replication group at the current site.

Exceptions

Table 54–5 GRANT_ADMIN_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REGISTER_USER_REPGROUP Procedure

This procedure assigns proxy materialized view administrator or receiver privileges at the master site or master materialized view site for use with remote sites. This procedure grants only the necessary privileges to the proxy materialized view administrator or receiver. It does not grant the powerful privileges granted by the GRANT_ADMIN_SCHEMA or GRANT_ADMIN_ANY_SCHEMA procedures.

Syntax

```
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
   username         IN   VARCHAR2,
   privilege_type   IN   VARCHAR2,
   {list_of_gnames  IN   VARCHAR2 |
   table_of_gnames  IN   DBMS_UTILITY.NAME_ARRAY)};
```

Note: This procedure is overloaded. The list_of_gnames and table_of_gnames parameters are mutually exclusive.

Parameters

Table 54–6 REGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user to whom you are giving either proxy materialized view administrator or receiver privileges.
privilege_type	Specifies the privilege type you are assigning. Use the following values for to define your privilege_type: ▪ receiver for receiver privileges ▪ proxy_snapadmin for proxy materialized view administration privileges
list_of_gnames	Comma-separated list of replication groups you want a user registered for receiver privileges. There must be no spaces between entries in the list. If you set list_of_gnames to NULL, then the user is registered for all replication groups, even replication groups that are not yet known when this procedure is called. You must use named notation in order to set list_of_gnames to NULL. An invalid replication group in the list causes registration to fail for the entire list.
table_of_gnames	PL/SQL index-by table of replication groups you want a user registered for receiver privileges. The PL/SQL index-by table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based (the positions start at 1 and increment by 1). Use the single value NULL to register the user for all replication groups. An invalid replication group in the table causes registration to fail for the entire table.

Exceptions

Table 54–7 REGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified replication group does not exist or the invocation database is not a master site or master materialized view site.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

REVOKE_ADMIN_ANY_SCHEMA Procedure

This procedure revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_ANY_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_ANY_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 54–8 REVOKE_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 54–9 REVOKE_ADMIN_ANY_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REVOKE_ADMIN_SCHEMA Procedure

This procedure revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 54–10 REVOKE_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 54–11 REVOKE_ADMIN_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

UNREGISTER_USER_REPGROUP Procedure

This procedure revokes the privileges and roles from the proxy materialized view administrator or receiver that were granted by the REGISTER_USER_REPGROUP procedure.

Syntax

```
DBMS_REPCAT_ADMIN.UNREGISTER_USER_REPGROUP (
   username         IN   VARCHAR2,
   privilege_type   IN   VARCHAR2,
   {list_of_gnames  IN   VARCHAR2 |
   table_of_gnames  IN   DBMS_UTILITY.NAME_ARRAY)};
```

> **Note:** This procedure is overloaded. The list_of_gnames and table_of_gnames parameters are mutually exclusive.

Parameters

Table 54–12 UNREGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are unregistering.
privilege_type	Specifies the privilege type you are revoking. Use the following values for to define your privilege_type: ▪ receiver for receiver privileges ▪ proxy_snapadmin for proxy materialized view administration privileges
list_of_gnames	Comma-separated list of replication groups you want a user unregistered for receiver privileges. There must be no spaces between entries in the list. If you set list_of_gnames to NULL, then the user is unregistered for all replication groups registered. You must use named notation in order to set list_of_gnames to NULL. An invalid replication group in the list causes unregistration to fail for the entire list.
table_of_gnames	PL/SQL index-by table of replication groups you want a user unregistered for receiver privileges. The PL/SQL index-by table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based (the positions start at 1 and increment by 1). Use the single value NULL to unregister the user for all replication groups registered. An invalid replication group in the table causes unregistration to fail for the entire table.

Exceptions

Table 54–13 UNREGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified replication group does not exist or the invocation database is not a master site or master materialized view site.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

55

DBMS_REPCAT_INSTANTIATE

The DBMS_REPCAT_INSTANTIATE package instantiates deployment templates.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT_INSTANTIATE Subprograms

Summary of DBMS_REPCAT_INSTANTIATE Subprograms

Table 55–1 DBMS_REPCAT_INSTANTIATE Package Subprograms

Subprogram	Description
DROP_SITE_INSTANTIATION Procedure on page 55-2	Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
INSTANTIATE_OFFLINE Function on page 55-3	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.
INSTANTIATE_ONLINE Function on page 55-5	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.

DROP_SITE_INSTANTIATION Procedure

This procedure drops a template instantiation at a target site. This procedure removes all related metadata at the master site and disables the specified site from refreshing its materialized views. You must execute this procedure as the user who originally instantiated the template. To see who instantiated the template, query the ALL_REPCAT_TEMPLATE_SITES view.

Syntax

```
DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
     refresh_template_name  IN   VARCHAR2,
     site_name              IN   VARCHAR2);
```

Table 55–2 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
site_name	Identifies the master site where you want to drop the specified template instantiation.

INSTANTIATE_OFFLINE Function

This function generates a file at the master site that is used to create the materialized view environment at the remote materialized view site while offline. This generated file is an offline instantiation file and should be used at remote materialized view sites that are not able to remain connected to the master site for an extended amount of time.

This is an ideal solution when the remote materialized view site is a laptop. Use the packaging interface in the Replication Management tool to package the generated file and data into a single file that can be posted on an FTP site or loaded to a CD-ROM, floppy disk, and so on.

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including the Replication Management tool, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the USER_REPCAT_TEMP_OUTPUT view.

The user who executes this public function becomes the "registered" user of the instantiated template at the specified site.

Note: This function is used in performing an offline instantiation of a deployment template.

This function should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view). See these respective packages for more information on their usage.

See Also:

- *Oracle9i Replication*
- The Replication Management tool's online help

Syntax

```
DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE(
     refresh_template_name   IN   VARCHAR2,
```

```
site_name              IN   VARCHAR2,
runtime_parm_id        IN   NUMBER    := -1e-130,
next_date              IN   DATE      := SYSDATE,
interval               IN   VARCHAR2  := 'SYSDATE + 1',
use_default_gowner     IN   BOOLEAN   := true)
return NUMBER;
```

Table 55–3 INSTANTIATE_OFFLINE Function Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, specify the identification used when creating the runtime parameters (the identification was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	The next refresh date value to be used when creating the refresh group.
interval	The refresh interval to be used when creating the refresh group.
use_default_gowner	If true, then any materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 55–4 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
dupl_template_site	The deployment template has already been instantiated at the materialized view site. A deployment template can be instantiated only once at a particular materialized view site.
not_authorized	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 55–5 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<system-generated number>	Specifies the generated system number for the `output_id` when you select from the `USER_REPCAT_TEMP_OUTPUT` view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE Function

This function generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online. This generated script should be used at remote materialized view sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views).

The script generated by this function is stored in the `USER_REPCAT_TEMP_OUTPUT` temporary view and is used by several Oracle tools, including the Replication Management tool, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the `USER_REPCAT_TEMP_OUTPUT` view.

The user who executes this public function becomes the "registered" user of the instantiated template at the specified site.

See Also:

- *Oracle9i Replication*
- The Replication Management tool's online help

Syntax

```
DBMS_REPCAT_INSTANTIATE.INSTANTIATE_ONLINE(
     refresh_template_name   IN   VARCHAR2,
     site_name               IN   VARCHAR2,
     runtime_parm_id         IN   NUMBER    := -1e-130,
     next_date               IN   DATE      := SYSDATE,
     interval                IN   VARCHAR2  := 'SYSDATE + 1',
     use_default_gowner      IN   BOOLEAN   := true)
     return NUMBER;
```

Table 55–6 INSTANTIATE_ONLINE Function Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, specify the identification used when creating the runtime parameters (the identification was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.
use_default_gowner	If true, then any materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Table 55–7 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
dupl_template_site	The deployment template has already been instantiated at the materialized view site. A deployment template can be instantiated only once at a particular materialized view site.
not_authorized	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 55–8 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<*system-generated number*>	Specifies the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

56

DBMS_REPCAT_RGT

DBMS_REPCAT_RGT controls the maintenance and definition of refresh group templates.

This chapter discusses the following topics:

Summary of DBMS_REPCAT_RGT Subprograms

Table 56–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
ALTER_REFRESH_TEMPLATE Procedure on page 56-4	Allows the DBA to alter existing deployment templates.
ALTER_TEMPLATE_OBJECT Procedure on page 56-6	Alters objects that have been added to a specified deployment template.
ALTER_TEMPLATE_PARM Procedure on page 56-9	Allows the DBA to alter the parameters for a specific deployment template.
ALTER_USER_ AUTHORIZATION Procedure on page 56-11	Alters the contents of the DBA_REPCAT_USER_ AUTHORIZATIONS view.
ALTER_USER_PARM_VALUE Procedure on page 56-12	Changes existing parameter values that have been defined for a specific user.
COMPARE_TEMPLATES Function on page 56-15	Allows the DBA to compare the contents of two deployment templates.
COPY_TEMPLATE Function on page 56-16	Allows the DBA to copy a deployment template.
CREATE_OBJECT_FROM_ EXISTING Function on page 56-18	Creates a template object definition from existing database objects and adds it to a target deployment template.
CREATE_REFRESH_ TEMPLATE Function on page 56-20	Creates the deployment template, which allows the DBA to define the template name, private / public status, and target refresh group.
CREATE_TEMPLATE_OBJECT Function on page 56-22	Adds object definitions to a target deployment template container.
CREATE_TEMPLATE_PARM Function on page 56-25	Creates parameters for a specific deployment template to allow custom data sets to be created at the remote materialized view site.
CREATE_USER_ AUTHORIZATION Function on page 56-27	Authorizes specific users to instantiate private deployment templates.
CREATE_USER_PARM_VALUE Function on page 56-29	Predefines deployment template parameter values for specific users.
DELETE_RUNTIME_PARMS Procedure on page 56-31	Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Table 56–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
DROP_ALL_OBJECTS Procedure on page 56-32	Allows the DBA to drop all objects or specific object types from a deployment template.
DROP_ALL_TEMPLATE_ PARMS Procedure on page 56-33	Allows the DBA to drop template parameters for a specified deployment template.
DROP_ALL_TEMPLATE_SITES Procedure on page 56-34	Removes all entries from the DBA_REPCAT_TEMPLATE_ SITES view.
DROP_ALL_TEMPLATES Procedure on page 56-35	Removes all deployment templates at the site where the procedure is called.
DROP_ALL_USER_ AUTHORIZATIONS Procedure on page 56-35	Allows the DBA to drop all user authorizations for a specified deployment template.
DROP_ALL_USER_PARM_ VALUES Procedure on page 56-36	Drops user parameter values for a specific deployment template.
DROP_REFRESH_TEMPLATE Procedure on page 56-38	Drops a deployment template.
DROP_SITE_INSTANTIATION Procedure on page 56-39	Removes the target site from the DBA_REPCAT_ TEMPLATE_SITES view.
DROP_TEMPLATE_OBJECT Procedure on page 56-40	Removes a template object from a specific deployment template.
DROP_TEMPLATE_PARM Procedure on page 56-41	Removes an existing template parameter from the DBA_ REPCAT_TEMPLATE_PARMS view.
DROP_USER_ AUTHORIZATION Procedure on page 56-42	Removes a user authorization entry from the DBA_ REPCAT_USER_AUTHORIZATIONS view.
DROP_USER_PARM_VALUE Procedure on page 56-43	Removes a predefined user parameter value for a specific deployment template.
GET_RUNTIME_PARM_ID Function on page 56-44	Retrieves an identification to be used when defining a runtime parameter value.
INSERT_RUNTIME_PARMS Procedure on page 56-45	Defines runtime parameter values prior to instantiating a template.
INSTANTIATE_OFFLINE Function on page 56-47	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.

Table 56–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
INSTANTIATE_ONLINE Function on page 56-50	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.
LOCK_TEMPLATE_ EXCLUSIVE Procedure on page 52	Prevents users from reading or instantiating the template when a deployment template is being updated or modified.
LOCK_TEMPLATE_SHARED Procedure on page 56-53	Makes a specified deployment template read-only.

ALTER_REFRESH_TEMPLATE Procedure

This procedure allows the DBA to alter existing deployment templates. Alterations may include defining a new deployment template name, a new refresh group, or a new owner and changing the public / private status.

Syntax

```
DBMS_REPCAT_RGT.ALTER_REFRESH_TEMPLATE (
   refresh_template_name      IN   VARCHAR2,
   new_owner                  IN   VARCHAR2 := '-',
   new_refresh_group_name     IN   VARCHAR2 := '-',
   new_refresh_template_name  IN   VARCHAR2 := '-',
   new_template_comment       IN   VARCHAR2 := '-',
   new_public_template        IN   VARCHAR2 := '-',
   new_last_modified          IN   DATE := to_date('1', 'J'),
   new_modified_by            IN   NUMBER := -1e-130);
```

Parameters

Table 56–2 ALTER_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template that you want to alter.
new_owner	The name of the new deployment template owner. Do not specify a value to keep the current owner.
new_refresh_group_name	If necessary, use this parameter to specify a new refresh group name to which the template objects will be added. Do not specify a value to keep the current refresh group.
new_refresh_template_name	Use this parameter to specify a new deployment template name. Do not specify a value to keep the current deployment template name.
new_template_comment	New deployment template comments. Do not specify a value to keep the current template comment.
new_public_template	Determines whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private). Do not specify a value to keep the current value.
new_last_modified	Contains the date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
new_modified_by	Contains the name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 56–3 ALTER_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.
dupl_refresh_template	A template with the specified name already exists.

ALTER_TEMPLATE_OBJECT Procedure

This procedure alters objects that have been added to a specified deployment template. The most common changes are altering the object DDL and assigning the object to a different deployment template.

Changes made to the template are reflected only at new sites instantiating the deployment template. Remote sites that have already instantiated the template must re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT (
   refresh_template_name       IN   VARCHAR2,
   object_name                 IN   VARCHAR2,
   object_type                 IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_object_name             IN   VARCHAR2 := '-',
   new_object_type             IN   VARCHAR2 := '-',
   new_ddl_text                IN   CLOB  := '-',
   new_master_rollback_seg     IN   VARCHAR2 := '-',
   new_flavor_id               IN   NUMBER := -1e-130);
```

Parameters

Table 56–4 ALTER_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the object that you want to alter.
object_name	Name of the template object that you want to alter.
object_type	Type of object that you want to alter.
new_refresh_template_name	Name of the new deployment template to which you want to reassign this object. Do not specify a value to keep the object assigned to the current deployment template.
new_object_name	New name of the template object. Do not specify a value to keep the current object name.
new_object_type	If specified, then the new object type. Objects of the following type may be specified: SNAPSHOT — PROCEDURE INDEX — FUNCTION TABLE — PACKAGE VIEW — PACKAGE BODY SYNONYM — TRIGGER SEQUENCE — DATABASE LINK
new_ddl_text	New object DDL for specified object. Do not specify any new DDL text to keep the current object DDL.
new_master_rollback_seg	New master rollback segment for specified object. Do not specify a value to keep the current rollback segment.
new_flavor_id	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 56–5 ALTER_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_flavor_id	If you receive this exception, contact Oracle Support Services.
bad_object_type	Object type is specified incorrectly. See Table 56–4 for a list of valid object types.
miss_template_ object	Template object name specified is invalid or does not exist.
dupl_template_ object	New template name specified in the new_refresh_ template_name parameter already exists.

Usage Notes

Because the ALTER_TEMPLATE_OBJECT procedure utilizes a CLOB, you must use the DBMS_LOB package when using the ALTER_TEMPLATE_OBJECT procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_ TEMPLATE_OBJECT procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE MATERIALIZED VIEW mview_sales AS SELECT *
      FROM sales WHERE salesperson = :salesid and region_id = :region';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT(
      refresh_template_name => 'rgt_personnel',
      object_name => 'MVIEW_SALES',
      object_type => 'SNAPSHOT',
      new_ddl_text => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

ALTER_TEMPLATE_PARM Procedure

This procedure allows the DBA to alter the parameters for a specific deployment template. Alterations include renaming the parameter and redefining the default value and prompt string.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
   refresh_template_name       IN   VARCHAR2,
   parameter_name              IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_parameter_name          IN   VARCHAR2 := '-',
   new_default_parm_value      IN   CLOB := NULL,
   new_prompt_string           IN   VARCHAR2 := '-',
   new_user_override           IN   VARCHAR2 := '-');
```

Parameters

Table 56–6 ALTER_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter that you want to alter.
parameter_name	Name of the parameter that you want to alter.
new_refresh_template_name	Name of the deployment template that the specified parameter should be reassigned to (useful when you want to move a parameter from one template to another). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	New name of the template parameter. Do not specify a value to keep the current parameter name.
new_default_parm_value	New default value for the specified parameter. Do not specify a value to keep the current default value.
new_prompt_string	New prompt text for the specified parameter. Do not specify a value to keep the current prompt string.
new_user_override	Determines whether the user can override the default value if prompted during the instantiation process. The user is prompted if no user parameter value has been defined for this parameter. Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to prevent an override.

Exceptions

Table 56–7 ALTER_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
dupl_template_parm	Combination of new_refresh_template_name and new_parameter_name already exists.

Usage Notes

Because the ALTER_TEMPLATE_PARM procedure utilizes a CLOB, you must use the DBMS_LOB package when using the ALTER_TEMPLATE_PARM procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_TEMPLATE_PARM procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      new_default_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

ALTER_USER_AUTHORIZATION Procedure

This procedure alters the contents of the DBA_REPCAT_USER_AUTHORIZATIONS view. Specifically, you can change user / deployment template authorization assignments. This procedure is helpful, for example, if an employee is reassigned and requires the materialized view environment of another deployment template. The DBA simply assigns the employee the new deployment template and the user is authorized to instantiate the target template.

Syntax

```
DBMS_REPCAT_RGT.ALTER_USER_AUTHORIZATION (
   user_name                    IN   VARCHAR2,
   refresh_template_name        IN   VARCHAR2,
   new_user_name                IN   VARCHAR2 := '-',
   new_refresh_template_name    IN   VARCHAR2 := '-');
```

Parameters

Table 56–8 ALTER_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
user_name	Name of the user whose authorization you want to alter.
refresh_template_name	Name of the deployment template that is currently assigned to the specified user that you want to alter.
new_user_name	Use this parameter to define a new user for this template authorization. Do not specify a value to keep the current user.
new_refresh_template_name	The deployment template that the specified user (either the existing or, if specified, the new user) is authorized to instantiate. Do not specify a value to keep the current deployment template.

Exceptions

Table 56–9 ALTER_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user_authorization	The combination of user_name and refresh_template_name values specified does not exist in the DBA_REPCAT_USER_AUTHORIZATIONS view.
miss_user	The user name specified for the new_user_name or user_name parameter is invalid or does not exist.
miss_refresh_template	The deployment template specified for the new_refresh_template parameter is invalid or does not exist.
dupl_user_authorization	A row already exists for the specified user name and deployment template name.

ALTER_USER_PARM_VALUE Procedure

This procedure changes existing parameter values that have been defined for a specific user. This procedure is especially helpful if your materialized view environment uses assignment tables. Change a user parameter value to quickly and securely change the data set of a remote materialized view site.

See Also: *Oracle9i Replication* for more information on using assignment tables

Syntax

```
DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
   refresh_template_name         IN   VARCHAR2,
   parameter_name                IN   VARCHAR2,
   user_name                     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2 := '-',
   new_parameter_name            IN   VARCHAR2 := '-',
   new_user_name                 IN   VARCHAR2 := '-',
   new_parm_value                IN   CLOB := NULL);
```

Parameters

Table 56–10 ALTER_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user parameter value that you want to alter.
parameter_name	Name of the parameter that you want to alter.
user_name	Name of the user whose parameter value you want to alter.
new_refresh_template_name	Name of the deployment template that the specified user parameter value should be reassigned to (useful when you are authorizing a user for a different template). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	The new template parameter name. Do not specify a value to keep the user value defined for the existing parameter.
new_user_name	The new user name that this parameter value is for. Do not specify a value to keep the parameter value assigned to the current user.
new_parm_value	The new parameter value for the specified user parameter. Do not specify a value to keep the current parameter value.

Exceptions

Table 56–11 ALTER_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
miss_user	User name specified for the user_name or new_user_name parameters is invalid or does not exist.
miss_user_parm_values	User parameter value specified does not exist.
dupl_user_parm_values	New user parameter specified already exists.

Usage Notes

Because the ALTER_USER_PARM_VALUE procedure utilizes a CLOB, you must use the DBMS_LOB package when using the ALTER_USER_PARM_VALUE procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_USER_PARM_VALUE procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      user_name => 'BOB',
      new_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

COMPARE_TEMPLATES Function

This function allows a DBA to compare the contents of two deployment templates. Any discrepancies between the two deployment templates is stored in the USER_REPCAT_TEMP_OUTPUT temporary view.

The COMPARE_TEMPLATES function returns a number that you specify in the WHERE clause when querying the USER_REPCAT_TEMP_OUTPUT temporary view. For example, if the COMPARE_TEMPLATES procedure returns the number 10, you would execute the following SELECT statement to view all discrepancies between two specified templates (your SELECT statement returns no rows if the templates are identical):

```
SELECT TEXT FROM USER_REPCAT_TEMP_OUTPUT
   WHERE OUTPUT_ID =  10 ORDER BY LINE;
```

The contents of the USER_REPCAT_TEMP_OUTPUT temporary view are lost after you disconnect or a rollback has been performed.

Syntax

```
DBMS_REPCAT_RGT.COMPARE_TEMPLATES (
   source_template_name    IN   VARCHAR2,
   compare_template_name   IN   VARCHAR2)
  return NUMBER;
```

Parameters

Table 56–12 COMPARE_TEMPLATES Function Parameters

Parameter	Description
source_template_name	Name of the first deployment template to be compared.
compare_template_name	Name of the second deployment template to be compared.

Exceptions

Table 56–13 COMPARE_TEMPLATES Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name to be compared is invalid or does not exist.

Returns

Table 56–14 COMPARE_TEMPLATES Function Returns

Return Value	Description
<system-generated number>	Specifies the number returned for the output_id value when you select from the USER_REPCAT_TEMP_OUTPUT temporary view to view the discrepancies between the compared templates.

COPY_TEMPLATE Function

This function enables you to copy a deployment template and is helpful when a new deployment template uses many of the objects contained in an existing deployment template. This function copies the deployment template, template objects, template parameters, and user parameter values. The DBA can optionally

have the function copy the user authorizations for this template. The number returned by this function is used internally by Oracle to manage deployment templates.

> **Note:** The values in the DBA_REPCAT_TEMPLATE_SITES view are not copied.

This function also allows the DBA to copy a deployment template to another master site, which is helpful for deployment template distribution and to split network loads between multiple sites.

Syntax

```
DBMS_REPCAT_RGT.COPY_TEMPLATE (
   old_refresh_template_name     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2,
   copy_user_authorizations      IN   VARCHAR2,
   dblink                        IN   VARCHAR2 := NULL)
  return NUMBER;
```

Parameters

Table 56–15 COPY_TEMPLATE Function Parameters

Parameter	Description
old_refresh_template_name	Name of the deployment template to be copied.
new_refresh_template_name	Name of the new deployment template.
copy_user_authorizations	Specifies whether the template authorizations for the original template should be copied for the new deployment template. Valid values for this parameter are Y, N, and NULL. **Note**: All users must exist at the target database.
dblink	Optionally defines where the deployment template should be copied from (this is helpful to distribute deployment templates to other master sites). If none is specified, then the deployment template is copied from the local master site.

Exceptions

Table 56–16 COPY_TEMPLATE Function Exceptions

Exception	Description
miss_refresh_ template	Deployment template name to be copied is invalid or does not exist.
dupl_refresh_ template	Name of the new refresh template specified already exists.
bad_copy_auth	Value specified for the copy_user_authorization parameter is invalid. Valid values are Y, N, and NULL.

Returns

Table 56–17 COPY_TEMPLATE Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

CREATE_OBJECT_FROM_EXISTING Function

This function creates a template object definition from existing database objects and adds it to a target deployment template. The object DDL that created the original database object is executed when the target deployment template is instantiated at the remote materialized view site. This is ideal for adding existing triggers and procedures to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_OBJECT_FROM_EXISTING(
     refresh_template_name  IN   VARCHAR2,
     object_name            IN   VARCHAR2,
     sname                  IN   VARCHAR2,
     oname                  IN   VARCHAR2,
     otype                  IN   VARCHAR2)
     return NUMBER;
```

Parameters

Table 56–18 CREATE_OBJECT_FROM_EXISTING Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to which you want to add this object.
object_name	Optionally, the new name of the existing object that you are adding to your deployment template (enables you to define a new name for an existing object).
sname	The schema that contains the object that you are creating your template object from.
oname	Name of the object that you are creating your template object from.
otype	The type of database object that you are adding to the template (that is, PROCEDURE, TRIGGER, and so on). The object type must be specified using the following numerical identifiers (DATABASE LINK, MATERIALIZED VIEW, and SNAPSHOT are not a valid object types for this function): SEQUENCE PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER

Exceptions

Table 56–19 CREATE_OBJECT_FROM_EXISTING Function Exceptions

Exception	Description
miss_refresh_template	The specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of existing deployment templates.
bad_object_type	The object type is specified incorrectly.
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.
objectmissing	The object specified does not exist.

Returns

Table 56–20 CREATE_OBJECT_FROM_EXISTING Function Returns

Return Value	Description
<*system-generated number*>	System-generated number used internally by Oracle.

CREATE_REFRESH_TEMPLATE Function

This function creates the deployment template, which enables you to define the template name, private / public status, and target refresh group. Each time that you create a template object, user authorization, or template parameter, you reference the deployment template created with this function. This function adds a row to the DBA_REPCAT_REFRESH_TEMPLATES view. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
   owner                  IN   VARCHAR2,
   refresh_group_name     IN   VARCHAR2,
   refresh_template_name  IN   VARCHAR2,
   template_comment       IN   VARCHAR2 := NULL,
   public_template        IN   VARCHAR2 := NULL,
   last_modified          IN   DATE := SYSDATE,
   modified_by            IN   VARCHAR2 := USER,
   creation_date          IN   DATE := SYSDATE,
   created_by             IN   VARCHAR2 := USER)
  return NUMBER;
```

Parameters

Table 56–21 CREATE_REFRESH_TEMPLATE Function Parameters

Parameter	Description
owner	User name of the deployment template owner is specified with this parameter. If an owner is not specified, then the name of the user creating the template is automatically used.
refresh_group_name	Name of the refresh group that is created when this template is instantiated. All objects created by this template are assigned to the specified refresh group.
refresh_template_name	Name of the deployment template that you are creating. This name is referenced in all activities that involve this deployment template.
template_comment	User comments defined with this parameter are listed in the DBA_REPCAT_REFRESH_TEMPLATES view.
public_template	Specifies whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private).
last_modified	The date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
modified_by	Name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.
creation_date	The date that this deployment template was created. If a value is not specified, then the current date is automatically used.
created_by	Name of the user who created this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 56–22 CREATE_REFRESH_TEMPLATE Function Exceptions

Exception	Description
dupl_refresh_ template	A template with the specified name already exists.
bad_public_ template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.

Returns

Table 56–23 CREATE_REFRESH_TEMPLATE Function Returns

Return Value	Description
<*system-generated number*>	System-generated number used internally by Oracle.

CREATE_TEMPLATE_OBJECT Function

This function adds object definitions to a target deployment template container. The specified object DDL is executed when the target deployment template is instantiated at the remote materialized view site. In addition to adding materialized views, this function can add tables, procedures, and other objects to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2,
   ddl_text               IN   CLOB,
   master_rollback_seg    IN   VARCHAR2 := NULL,
   flavor_id              IN   NUMBER := -1e-130)
  return NUMBER;
```

Parameters

Table 56–24 CREATE_TEMPLATE_OBJECT Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to which you want to add this object.
object_name	Name of the template object that you are creating.
object_type	The type of database object that you are adding to the template (that is, SNAPSHOT, TRIGGER, PROCEDURE, and so on). Objects of the following type may be specified: SNAPSHOT — PROCEDURE INDEX — FUNCTION TABLE — PACKAGE VIEW — PACKAGE BODY SYNONYM — TRIGGER SEQUENCE — DATABASE LINK
ddl_text	Contains the DDL that creates the object that you are adding to the template. Be sure to end your DDL with a semi-colon. You can use a colon (:) to create a template parameter for your template object. When you add a materialized view (snapshot) with a CREATE MATERIALIZED VIEW statement, make sure you specify the schema name of the owner of the master table in the materialized view query.
master_rollback_seg	Specifies the name of the rollback segment to use when executing the defined object DDL at the remote materialized view site.
flavor_id	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 56–25 CREATE_TEMPLATE_OBJECT Function Exceptions

Exception	Description
`miss_refresh_template`	Specified refresh template name is invalid or missing. Query the `DBA_REPCAT_REFRESH_TEMPLATES` view for a list of existing deployment templates.
`bad_object_type`	Object type is specified incorrectly. See Table 56–24 for a list of valid object types.
`dupl_template_object`	An object of the same name and type has already been added to the specified deployment template.

Returns

Table 56–26 CREATE_TEMPLATE_OBJECT Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

Usage Notes

Because CREATE_TEMPLATE_OBJECT utilizes a CLOB, you must use the DBMS_LOB package when using the CREATE_TEMPLATE_OBJECT function. The following example illustrates how to use the DBMS_LOB package with the CREATE_TEMPLATE_OBJECT function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE MATERIALIZED VIEW mview_sales AS SELECT *
        FROM sales WHERE salesperson = :salesid';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT(
        refresh_template_name => 'rgt_personnel',
        object_name => 'mview_sales',
        object_type => 'SNAPSHOT',
        ddl_text => templob,
        master_rollback_seg => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

CREATE_TEMPLATE_PARM Function

This function creates parameters for a specific deployment template to allow custom data sets to be created at the remote materialized view site. This function is only required when the DBA wants to define a set of template variables before adding any template objects. When objects are added to the template using the CREATE_TEMPLATE_OBJECT function, any variables in the object DDL are automatically added to the DBA_REPCAT_TEMPLATE_PARMS view.

The DBA typically uses the ALTER_TEMPLATE_PARM function to modify the default parameter values or prompt strings (see "ALTER_TEMPLATE_PARM Procedure" on page 56-9 for more information). The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
   parameter_name         IN   VARCHAR2,
   default_parm_value     IN   CLOB := NULL,
   prompt_string          IN   VARCHAR2 := NULL,
   user_override          IN   VARCHAR2 := NULL)
   return NUMBER;
```

Parameters

Table 56–27 CREATE_TEMPLATE_PARM Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template for which you want to create the parameter.
parameter_name	Name of the parameter you are creating.
default_parm_value	Default values for this parameter are defined using this parameter. If a user parameter value or runtime parameter value is not present, then this default value is used during the instantiation process.
prompt_string	The descriptive prompt text that is displayed for this template parameter during the instantiation process.
user_override	Determines whether the user can override the default value if prompted during the instantiation process. The user is prompted if no user parameter value has been defined for this parameter. Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to not allow an override.

Exceptions

Table 56–28 CREATE_TEMPLATE_PARM Function Exceptions

Exception	Description
miss_refresh_template	The specified refresh template name is invalid or missing.
dupl_template_parm	A parameter of the same name has already been defined for the specified deployment template.

Returns

Table 56–29 CREATE_TEMPLATE_PARM Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

Usage Notes

Because the CREATE_TEMPLATE_PARM function utilizes a CLOB, you must use the DBMS_LOB package when using the CREATE_TEMPLATE_PARM function. The following example illustrates how to use the DBMS_LOB package with the CREATE_TEMPLATE_PARM function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_PARM(
        refresh_template_name => 'rgt_personnel',
        parameter_name => 'region',
        default_parm_value => templob,
        prompt_string => 'Enter your region ID:',
        user_override => 'Y');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

CREATE_USER_AUTHORIZATION Function

This function authorizes specific users to instantiate private deployment templates. Users not authorized for a private deployment template are not able to instantiate the private template. This function adds a row to the DBA_REPCAT_USER_AUTHORIZATIONS view.

Before you authorize a user, verify that the user exists at the master site where the user will instantiate the deployment template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION (
   user_name               IN   VARCHAR2,
   refresh_template_name   IN   VARCHAR2)
   return NUMBER;
```

Parameters

Table 56–30 CREATE_USER_AUTHORIZATION Function Parameters

Parameter	Description
user_name	Name of the user that you want to authorize to instantiate the specified template. Specify multiple users by separating user names with a comma (for example, 'john, mike, bob')
refresh_template_name	Name of the template that you want to authorize the specified user to instantiate.

Exceptions

Table 56–31 CREATE_USER_AUTHORIZATION Function Exceptions

Exception	Description
miss_user	User name supplied is invalid or does not exist.
miss_refresh_template	Refresh template name supplied is invalid or does not exist.
dupl_user_authorization	An authorization has already been created for the specified user and deployment template.

Returns

Table 56–32 CREATE_USER_AUTHORIZATION Function Returns

Return Value	Description
<*system-generated number*>	System-generated number used internally by Oracle.

CREATE_USER_PARM_VALUE Function

This function predefines deployment template parameter values for specific users. For example, if you want to predefine the region parameter as `west` for user `33456`, then you would use the this function.

Any values specified with this function take precedence over default values specified for the template parameter. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
   refresh_template_name    IN   VARCHAR2,
   parameter_name           IN   VARCHAR2,
   user_name                IN   VARCHAR2,
   parm_value               IN   CLOB := NULL)
  return NUMBER;
```

Parameters

Table 56–33 CREATE_USER_PARM_VALUE Function Parameters

Parameter	Description
`refresh_template_name`	Specifies the name of the deployment template that contains the parameter you are creating a user parameter value for.
`parameter_name`	Name of the template parameter that you are defining a user parameter value for.
`user_name`	Specifies the name of the user that you are predefining a user parameter value for.
`parm_value`	The predefined parameter value that will be used during the instantiation process initiated by the specified user.

Exceptions

Table 56–34 CREATE_USER_PARM_VALUE Function Exceptions

Exception	Description
miss_refresh_ template	Specified deployment template name is invalid or missing.
dupl_user_parm_ values	A parameter value for the specified user, parameter, and deployment template has already been defined. Query the DBA_ REPCAT_USER_PARM_VALUES view for a listing of existing user parameter values.
miss_template_parm	Specified deployment template parameter name is invalid or missing.
miss_user	Specified user name is invalid or missing.

Returns

Table 56–35 CREATE_USER_PARM_VALUE Function Returns

Return Value	Description
<*system-generated number*>	System-generated number used internally by Oracle.

Usage Notes

Because the CREATE_USER_PARM_VALUE function utilizes a CLOB, you must use the DBMS_LOB package when using the this function. The following example illustrates how to use the DBMS_LOB package with the CREATE_USER_PARM_VALUE function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE(
        refresh_template_name => 'rgt_personnel',
        parameter_name => 'region',
        user_name => 'BOB',
        user_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

DELETE_RUNTIME_PARMS Procedure

Use this procedure before instantiating a deployment template to delete a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Syntax

```
DBMS_REPCAT_RGT.DELETE_RUNTIME_PARMS(
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2);
```

Parameters

Table 56–36 DELETE_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	Specifies the identification that you previously assigned the runtime parameter value to (this value was retrieved using the GET_RUNTIME_PARM_ID function).
parameter_name	Specifies the name of the parameter value that you want to drop (query the DBA_REPCAT_TEMPLATE_PARMS view for a list of deployment template parameters).

Exceptions

Table 56–37 DELETE_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_template_parm	The specified deployment template parameter name is invalid or missing.

DROP_ALL_OBJECTS Procedure

This procedure allows the DBA to drop all objects or specific object types from a deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_OBJECTS (
   refresh_template_name   IN   VARCHAR2,
   object_type             IN   VARCHAR2 := NULL);
```

Parameters

Table 56–38 DROP_ALL_OBJECTS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.
object_type	If NULL, then all objects in the template are dropped. If an object type is specified, then only objects of that type are dropped. Objects of the following type may be specified: SNAPSHOT — PROCEDURE INDEX — FUNCTION TABLE — PACKAGE VIEW — PACKAGE BODY SYNONYM — TRIGGER SEQUENCE — DATABASE LINK

Exceptions

Table 56–39 DROP_ALL_OBJECTS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 56–38 for a list of valid object types.

DROP_ALL_TEMPLATE_PARMS Procedure

This procedure lets you drop template parameters for a specified deployment template. You can use this procedure to drop all parameters that are not referenced by a template object or to drop from the template all objects that reference any parameter, along with all of the parameters themselves.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_PARMS (
   refresh_template_name   IN   VARCHAR2,
   drop_objects            IN   VARCHAR2 := 'n');
```

Parameters

Table 56–40 DROP_ALL_TEMPLATE_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameters and objects that you want to drop.
drop_objects	If no value is specified, then this parameter defaults to N, which drops all parameters not referenced by a template object. If Y is specified, then all objects that reference any template parameter and the template parameters themselves are dropped. The objects are dropped from the template, not from the database.

Exceptions

Table 56–41 DROP_ALL_TEMPLATE_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATE_SITES Procedure

This procedure removes all entries from the DBA_REPCAT_TEMPLATE_SITES view, which keeps a record of sites that have instantiated a particular deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_SITES (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 56–42 DROP_ALL_TEMPLATE_SITES Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the sites that you want to drop.

Exceptions

Table 56–43 DROP_ALL_TEMPLATE_SITES Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATES Procedure

This procedure removes all deployment templates at the site where the procedure is called.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATES;
```

DROP_ALL_USER_AUTHORIZATIONS Procedure

This procedure enables the DBA to drop all user authorizations for a specified deployment template. Executing this procedure removes rows from the DBA_REPCAT_USER_AUTHORIZATIONS view.

This procedure might be implemented after converting a private template to a public template and the user authorizations are no longer required.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_AUTHORIZATIONS (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 56–44 DROP_ALL_USER_AUTHORIZATIONS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user authorizations that you want to drop.

Exceptions

Table 56–45 DROP_ALL_USER_AUTHORIZATIONS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_USER_PARM_VALUES Procedure

This procedure drops user parameter values for a specific deployment template. This procedure is very flexible and enables you to define a set of user parameter values to be deleted. For example, defining the following parameters has the effect described in Table 56–46.

Table 56–46 DROP_ALL_USER_PARM_VALUES Procedure

Parameter	Effect
refresh_template_name	Drops all user parameters for the specified deployment template
refresh_template_name and user_name	Drops all of the specified user parameters for the specified deployment template
refresh_template_name and parameter_name	Drops all user parameter values for the specified deployment template parameter

Table 56–46 DROP_ALL_USER_PARM_VALUES Procedure

Parameter	Effect
refresh_template_name, parameter_name, and user_name	Drops the specified user's value for the specified deployment template parameter (equivalent to drop_user_parm)

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_PARMS (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2,
   parameter_name          IN   VARCHAR2);
```

Parameters

Table 56–47 DROP_ALL_USER_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter values that you want to drop.
user_name	Name of the user whose parameter values you want to drop.
parameter_name	Template parameter that contains the values that you want to drop.

Exceptions

Table 56–48 DROP_ALL_USER_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.

DROP_REFRESH_TEMPLATE Procedure

This procedure drops a deployment template. Dropping a deployment template has a cascading effect, removing all related template parameters, user authorizations, template objects, and user parameters (this procedure does not drop template sites).

Syntax

```
DBMS_REPCAT_RGT.DROP_REFRESH_TEMPLATE (
   refresh_template_name  IN   VARCHAR2);
```

Parameters

Table 56–49 DROP_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be dropped.

Exceptions

Table 56–50 DROP_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of deployment templates.

DROP_SITE_INSTANTIATION Procedure

This procedure drops a template instantiation at any target site. This procedure removes all related metadata at the master site and disables the specified site from refreshing its materialized views.

Syntax

```
DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
     refresh_template_name  IN   VARCHAR2,
     user_name              IN   VARCHAR2,
     site_name              IN   VARCHAR2);
```

Table 56–51 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
user_name	The name of the user who originally instantiated the template at the remote materialized view site. Query the ALL_REPCAT_TEMPLATE_SITES view to see the users that instantiated templates.

Table 56–51 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
site_name	Identifies the master site where you want to drop the specified template instantiation.

Exceptions

Table 56–52 DROP_SITE_INSTANTIATION Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_user	The username specified does not exist.
miss_template_site	The deployment template has not been instantiated for user and site.

DROP_TEMPLATE_OBJECT Procedure

This procedure removes a template object from a specific deployment template. For example, a DBA would use this procedure to remove an outdated materialized view from a deployment template. Changes made to the template are reflected at new sites instantiating the deployment template. Remote sites that have already instantiated the template must re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2);
```

Parameters

Table 56–53 DROP_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description	
refresh_template_name	Name of the deployment template from which you are dropping the object.	
object_name	Name of the template object to be dropped.	
object_type	The type of object that is to be dropped. Objects of the following type may be specified:	
	SNAPSHOT	PROCEDURE
	INDEX	FUNCTION
	TABLE	PACKAGE
	VIEW	PACKAGE BODY
	SYNONYM	TRIGGER
	SEQUENCE	DATABASE LINK

Exceptions

Table 56–54 DROP_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_template_object	The template object specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_OBJECTS view to see a list of deployment template objects.

DROP_TEMPLATE_PARM Procedure

This procedure removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view. This procedure is useful when you have dropped a template object and a particular parameter is no longer needed.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
```

```
   parameter_name          IN   VARCHAR2);
```

Parameters

Table 56–55 DROP_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	The deployment template name that has the parameter that you want to drop
parameter_name	Name of the parameter that you want to drop.

Exceptions

Table 56–56 DROP_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_template_parm	The parameter name specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_PARMS view to see a list of template parameters.

DROP_USER_AUTHORIZATION Procedure

This procedure removes a user authorization entry from the DBA_REPCAT_USER_AUTHORIZATIONS view. This procedure is used when removing a user's template authorization. If a user's authorization is removed, then the user is no longer able to instantiate the target deployment template.

> **See Also:** "DROP_ALL_USER_AUTHORIZATIONS Procedure" on page 56-35

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2);
```

Parameters

Table 56–57 DROP_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template from which the user's authorization is being removed.
user_name	Name of the user whose authorization is being removed.

Exceptions

Table 56–58 DROP_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user	Specified user name is invalid or does not exist.
miss_user_authorization	Specified user and deployment template combination does not exist. Query the DBA_REPCAT_USER_AUTHORIZATIONS view to see a list of user / deployment template authorizations.
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_USER_PARM_VALUE Procedure

This procedure removes a predefined user parameter value for a specific deployment template. This procedure is often executed after a user's template authorization has been removed.

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_PARM_VALUE (
   refresh_template_name    IN   VARCHAR2,
   parameter_name           IN   VARCHAR2,
   user_name                IN   VARCHAR2);
```

Parameters

Table 56–59 DROP_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the parameter value that you want to drop.
parameter_name	Parameter name that contains the predefined value that you want to drop.
user_name	Name of the user whose parameter value you want to drop.

Exceptions

Table 56–60 DROP_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.

GET_RUNTIME_PARM_ID Function

This function retrieves an identification to be used when defining a runtime parameter value. All runtime parameter values are assigned to this identification and are also used during the instantiation process.

Syntax

```
DBMS_REPCAT_RGT.GET_RUNTIME_PARM_ID
  RETURN NUMBER;
```

Returns

Table 56–61 GET_RUNTIME_PARM_ID Function Returns

Return Value	Corresponding Datatype
<system-generated number>	Runtime parameter values are assigned to the system-generated number and are also used during the instantiation process.

INSERT_RUNTIME_PARMS Procedure

This procedure defines runtime parameter values prior to instantiating a template. This procedure should be used to define parameter values when no user parameter values have been defined and you do not want to accept the default parameter values.

Before using the this procedure, be sure to execute the GET_RUNTIME_PARM_ID function to retrieve a parameter identification to use when inserting a runtime parameter. This identification is used for defining runtime parameter values and instantiating deployment templates.

Syntax

```
DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS (
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2,
   parameter_value    IN   CLOB);
```

Parameters

Table 56–62 INSERT_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	The identification retrieved by the GET_RUNTIME_PARM_ID function. This identification is also used when instantiating the deployment template. Be sure to use the same identification for all parameter values for a deployment template.
parameter_name	Name of the template parameter for which you are defining a runtime parameter value. Query the DBA_REPCAT_TEMPLATE_PARMS view for a list of template parameters.

Table 56–62 INSERT_RUNTIME_PARMS Procedure Parameters

Parameter	Description
parameter_value	The runtime parameter value that you want to use during the deployment template instantiation process.

Exceptions

Table 56–63 INSERT_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_user	The user name specified is invalid or does not exist.
miss_user_parm_values	The deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.

Usage Notes

Because the this procedure utilizes a CLOB, you must use the DBMS_LOB package when using the INSERT_RUNTIME_PARMS procedure. The following example illustrates how to use the DBMS_LOB package with the INSERT_RUNTIME_PARMS procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS(
      runtime_parm_id => 20,
      parameter_name => 'region',
      parameter_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

INSTANTIATE_OFFLINE Function

This function generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while the materialized view site disconnected from the master (that is, while the materialized view site is offline). This generated script should be used at remote materialized view sites that are not able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views). This function must be executed separately for each user instantiation.

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the USER_REPCAT_TEMP_OUTPUT temporary view.

Note: This function is used to perform an offline instantiation of a deployment template. Additionally, this function is for replication administrators who are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the INSTANTIATE_OFFLINE function. See the "INSTANTIATE_OFFLINE Function" on page 56-47 for more information.

This function should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view). See these respective packages for more information on their usage.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
     refresh_template_name   IN   VARCHAR2,
     site_name               IN   VARCHAR2,
     user_name               IN   VARCHAR2  := NULL,
     runtime_parm_id         IN   NUMBER    := -1e-130,
     next_date               IN   DATE      := SYSDATE,
     interval                IN   VARCHAR2  := 'SYSDATE + 1',
     use_default_gowner      IN   BOOLEAN   := true)
     return NUMBER;
```

Parameters

Table 56–64 INSTANTIATE_OFFLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user who is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the identification used when creating the runtime parameters (the identification was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.
use_default_gowner	If true, then any materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 56–65 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_USER_AUTHORIZATIONS view. If user is not listed, then the specified user is not authorized to instantiate the target deployment template.

Returns

Table 56–66 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<*system-generated number*>	Specifies the generated system number for the `output_id` when you select from the `USER_REPCAT_TEMP_OUTPUT` temporary view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE Function

This function generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while the materialized view site is connected to the master (that is, while the materialized view site is online). This generated script should be used at remote materialized view sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views). This function must be executed separately for each user instantiation.

The script generated by this function is stored in the `USER_REPCAT_TEMP_OUTPUT` temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the `USER_REPCAT_TEMP_OUTPUT` temporary view.

Note: This function is for replication administrators who are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the `INSTANTIATE_OFFLINE` function, described in "INSTANTIATE_OFFLINE Function" on page 56-47 section.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
   refresh_template_name   IN   VARCHAR2,
   site_name               IN   VARCHAR2  := NULL,
   user_name               IN   VARCHAR2  := NULL,
   runtime_parm_id         IN   NUMBER    := -1e-130,
   next_date               IN   DATE      := SYSDATE,
   interval                IN   VARCHAR2  := 'SYSDATE + 1',
   use_default_gowner      IN   BOOLEAN   := true)
   return NUMBER;
```

Parameters

Table 56–67 INSTANTIATE_ONLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user who is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the identification used when creating the runtime parameters (the identification was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.
use_default_gowner	If true, then any materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 56–68 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_ template	Specified deployment template name is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_USER_ AUTHORIZATIONS view. If user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	Not all of the template parameters were populated by the defined user parameter values or template default values. The number of predefined values may not have matched the number of template parameters or a predefined value was invalid for the target parameter (that is, type mismatch).

Returns

Table 56–69 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<*system-generated number*>	Specifies the system-generated number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT temporary view to retrieve the generated instantiation script.

LOCK_TEMPLATE_EXCLUSIVE Procedure

When a deployment template is being updated or modified, you should use the LOCK_TEMPLATE_EXCLUSIVE procedure to prevent users from reading or instantiating the template.

The lock is released when a ROLLBACK or COMMIT is performed.

Note: This procedure should be executed before you make any modifications to your deployment template.

Syntax

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_EXCLUSIVE();
```

LOCK_TEMPLATE_SHARED Procedure

The LOCK_TEMPLATE_SHARED procedure is used to make a specified deployment template "read-only." This procedure should be called before instantiating a template, as this ensures that nobody can change the deployment template while it is being instantiated.

The lock is released when a ROLLBACK or COMMIT is performed.

Syntax

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_SHARED();
```

57

DBMS_REPUTIL

DBMS_REPUTIL contains subprograms to generate shadow tables, triggers, and packages for table replication, as well as subprograms to generate wrappers for replication of standalone procedure invocations and packaged procedure invocations. This package is referenced only by the generated code.

This chapter discusses the following topics:

- Summary of DBMS_REPUTIL Subprograms

Summary of DBMS_REPUTIL Subprograms

Table 57–1 DBMS_REPUTIL Package Subprograms

Subprogram	Description
REPLICATION_OFF Procedure on page 57-2	Modifies tables without replicating the modifications to any other sites in the replication environment, or disables row-level replication when using procedural replication.
REPLICATION_ON Procedure on page 57-3	Re-enables replication of changes after replication has been temporarily suspended.
REPLICATION_IS_ON Function on page 57-3	Determines whether or not replication is running.
FROM_REMOTE Function on page 57-3	Returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures.
GLOBAL_NAME Function on page 57-4	Determines the global database name of the local database (the global name is the returned value).
MAKE_INTERNAL_ PKG Procedure on page 57-4	Synchronizes internal packages and tables in the replication catalog. **Note:** Do not execute this procedure unless directed to do so by Oracle Support Services.
SYNC_UP_REP Procedure on page 57-5	Synchronizes internal triggers and tables / materialized views in the replication catalog. **Note:** Do not execute this procedure unless directed to do so by Oracle Support Services.

REPLICATION_OFF Procedure

This procedure enables you to modify tables without replicating the modifications to any other sites in the replication environment. It also disables row-level replication when using procedural replication. In general, you should suspend replication activity for all master groups in your replication environment before setting this flag.

Syntax

```
DBMS_REPUTIL.REPLICATION_OFF();
```

REPLICATION_ON Procedure

This procedure re-enables replication of changes after replication has been temporarily suspended.

Syntax

```
DBMS_REPUTIL.REPLICATION_ON();
```

REPLICATION_IS_ON Function

This function determines whether or not replication is running. A returned value of TRUE indicates that the generated replication triggers are enabled. A return value of FALSE indicates that replication is disabled at the current site for the replication group.

The returning value of this function is set by calling the REPLICATION_ON or REPLICATION_OFF procedures in the DBMS_REPUTIL package.

Syntax

```
DBMS_REPUTIL.REPLICATION_IS_ON()
  return BOOLEAN;
```

FROM_REMOTE Function

This function returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures. You may need to check this function if you have any triggers that could be fired as the result of an update by an internal package.

Syntax

```
DBMS_REPUTIL.FROM_REMOTE()
   return BOOLEAN;
```

GLOBAL_NAME Function

This function determines the global database name of the local database (the global name is the returned value).

Syntax

```
DBMS_REPUTIL.GLOBAL_NAME()
   return VARCHAR2;
```

MAKE_INTERNAL_PKG Procedure

This procedure synchronizes the existence of an internal package with a table or materialized view in the replication catalog. If the table has replication support, then execute this procedure to create the internal package. If replication support does not exist, then this procedure destroys any related internal package. This procedure does not accept the storage table of a nested table.

Caution: Do not execute this procedure unless directed to do so by Oracle Support Services.

Syntax

```
DBMS_REPUTIL.MAKE_INTERNAL_PKG (
   canon_sname     IN   VARCHAR2,
   canon_oname     IN   VARCHAR2);
```

Parameters

Table 57–2 MAKE_INTERNAL_PKG Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).

SYNC_UP_REP Procedure

This procedure synchronizes the existence of an internal trigger with a table or materialized view in the replication catalog. If the table or materialized view has replication support, then execute this procedure to create the internal replication trigger. If replication support does not exist, then this procedure destroys any related internal trigger. This procedure does not accept the storage table of a nested table.

Caution: Do not execute this procedure unless directed to do so by Oracle Support Services.

Syntax

```
DBMS_REPUTIL.SYNC_UP_REP (
   canon_sname    IN   VARCHAR2,
   canon_oname    IN   VARCHAR2);
```

Parameters

Table 57–3 SYNC_UP_REP Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table or materialized view to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table or materialized view to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).

58

DBMS_RESOURCE_MANAGER

The DBMS_RESOURCE_MANAGER package maintains plans, consumer groups, and plan directives. It also provides semantics so that you may group together changes to the plan schema.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle9i Database Administrator's Guide.*

This chapter discusses the following topics:

Requirements

The invoker must have the ADMINISTER_RESOURCE_MANAGER system privilege to execute these procedures. The procedures to grant and revoke this privilege are in the package DBMS_RESOURCE_MANAGER_PRIVS.

Summary of DBMS_RESOURE_MANAGER Subprograms

Table 58–1 DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
CREATE_PLAN Procedure on page 58-3	Creates entries which define resource plans.
CREATE_SIMPLE_PLAN Procedure on page 58-4	Creates a single-level resource plan containing up to eight consumer groups in one step.
UPDATE_PLAN Procedure on page 58-5	Updates entries which define resource plans.
DELETE_PLAN Procedure on page 58-6	Deletes the specified plan as well as all the plan directives it refers to.
DELETE_PLAN_CASCADE Procedure on page 58-6	Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups).
CREATE_CONSUMER_ GROUP Procedure on page 58-7	Creates entries which define resource consumer groups.
UPDATE_CONSUMER_ GROUP Procedure on page 58-8	Updates entries which define resource consumer groups.
DELETE_CONSUMER_ GROUP Procedure on page 58-8	Deletes entries which define resource consumer groups.
CREATE_PLAN_DIRECTIVE Procedure on page 58-9	Creates resource plan directives.
UPDATE_PLAN_DIRECTIVE Procedure on page 58-11	Updates resource plan directives.
DELETE_PLAN_DIRECTIVE Procedure on page 58-12	Deletes resource plan directives.
CREATE_PENDING_AREA Procedure on page 58-13	Creates a work area for changes to resource manager objects.

Table 58–1 DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
VALIDATE_PENDING_AREA Procedure on page 58-14	Validates pending changes for the resource manager.
CLEAR_PENDING_AREA Procedure on page 58-14	Clears the work area for the resource manager.
SUBMIT_PENDING_AREA Procedure on page 58-15	Submits pending changes for the resource manager.
SET_INITIAL_CONSUMER_ GROUP Procedure on page 58-18	Assigns the initial resource consumer group for a user.
SWITCH_CONSUMER_ GROUP_FOR_SESS Procedure on page 58-19	Changes the resource consumer group of a specific session.
SWITCH_CONSUMER_ GROUP_FOR_USER Procedure on page 58-20	Changes the resource consumer group for all sessions with a given user name.

CREATE_PLAN Procedure

This procedure creates entries which define resource plans. For release 8.2, max_ active_sess_target_mth was renamed active_sess_pool_mth and new_queueing_ mth was added.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
   plan                      IN VARCHAR2,
   comment                   IN VARCHAR2,
   cpu_mth                   IN VARCHAR2 DEFAULT 'EMPHASIS',
   active_sess_pool_mth       IN VARCHAR2 DEFAULT 'ACTIVE_SESS_POOL_ABSOLUTE',
   parallel_degree_limit_mth  IN VARCHAR2 DEFAULT
                             'PARALLEL_DEGREE_LIMIT_ABSOLUTE',
   queueing_mth              IN VARCHAR2 DEFAULT 'FIFO_TIMEOUT',);
```

Parameters

Table 58–2 CREATE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan.
comment	User's comment.
cpu_mth	Allocation method for CPU resources.
active_sess_pool_mth	Allocation method for maximum active sessions.
parallel_degree_limit_mth	Allocation method for degree of parallelism.
new_queueing_mth	Specifies type of queuing policy to use with active session pool feature.

CREATE_SIMPLE_PLAN Procedure

This procedure creates a single-level resource plan containing up to eight consumer groups in one step. You do not need to create a pending area manually before creating a resource plan, or use the CREATE_CONSUMER_GROUP and CREATE_RESOURCE_PLAN_DIRECTIVES procedures separately.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
   SIMPLE_PLAN      IN  VARCHAR2  DEFAULT,
   CONSUMER_GROUP1  IN  VARCHAR2  DEFAULT,
   GROUP1_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP2  IN  VARCHAR2  DEFAULT,
   GROUP2_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP3  IN  VARCHAR2  DEFAULT,
   GROUP3_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP4  IN  VARCHAR2  DEFAULT,
   GROUP4_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP5  IN  VARCHAR2  DEFAULT,
   GROUP5_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP6  IN  VARCHAR2  DEFAULT,
   GROUP6_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP7  IN  VARCHAR2  DEFAULT,
   GROUP7_CPU       IN  NUMBER    DEFAULT,
   CONSUMER_GROUP8  IN  VARCHAR2  DEFAULT,
```

```
   GROUP8_CPU        IN  NUMBER    DEFAULT);
```

UPDATE_PLAN Procedure

This procedure updates entries which define resource plans.For release 8.2 new_max_active_sess_target_mth was renamed new_active_sess_pool_mth and new_queueing_mth was added.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN (
   plan                              IN VARCHAR2,
   new_comment                       IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth                       IN VARCHAR2 DEFAULT NULL,
   new_active_sess_pool_mth          IN VARCHAR2 DEFAULT NULL,
   new_parallel_degree_limit_mth     IN VARCHAR2 DEFAULT NULL,
   new_queueing_mth                  IN VARCHAR2 DEFAULT NULL,
   new_group_switch_mth              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–3 UPDATE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan.
new_comment	New user's comment.
new_cpu_mth	Name of new allocation method for CPU resources.
new_active_sess_pool_mth	Name of new method for maximum active sessions.
new_parallel_degree_limit_mth	Name of new method for degree of parallelism.
new_queueing_mth	Specifies type of queuing policy to use with active seesion pool feature.

Usage Notes

If the parameters to UPDATE_PLAN are not specified, then they remain unchanged in the data dictionary.

DELETE_PLAN Procedure

This procedure deletes the specified plan as well as all the plan directives to which it refers.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN (
   plan IN VARCHAR2);
```

Parameters

Table 58–4 DELETE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan to delete.

DELETE_PLAN_CASCADE Procedure

This procedure deletes the specified plan and all of its descendants (plan directives, subplans, consumer groups). Mandatory objects and directives are not deleted.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (
   plan IN VARCHAR2);
```

Parameters

Table 58–5 DELETE_PLAN_CASCADE Procedure Parameters

Parameters	Description
plan	Name of plan.

Errors

If DELETE_PLAN_CASCADE encounters any error, then it rolls back, and nothing is deleted.

Note: If you want to use any default resource allocation method, then you do not need not specify it when creating or updating a plan.

Usage Notes

Defaults are:

- cpu_method = EMPHASIS
- parallel_degree_limit_mth = PARALLEL_DEGREE_LIMIT_ABSOLUTE
- active_sess_pool_mth = MAX_ACTIVE_SESS_ABSOLUTE

Note: The parameter max_active_sess_target_mth is undocumented in this release: It is reserved for future use.

CREATE_CONSUMER_GROUP Procedure

This procedure lets you create entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   consumer_group  IN VARCHAR2,
   comment         IN VARCHAR2,
   cpu_mth         IN VARCHAR2 DEFAULT 'ROUND-ROBIN');
```

Parameters

Table 58–6 CREATE_CONSUMER_GROUP Procedure Parameters

Parameter	Description
consumer_group	Name of consumer group.
comment	User's comment.
cpu_mth	Name of CPU resource allocation method.

UPDATE_CONSUMER_GROUP Procedure

This procedure lets you update entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP (
   consumer_group  IN VARCHAR2,
   new_comment     IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 58–7 UPDATE_CONSUMER_GROUP Procedure Parameter

Parameter	Description
consumer_group	Name of consumer group.
new_comment	New user's comment.
new_cpu_mth	Name of new method for CPU resource allocation.

If the parameters to the UPDATE_CONSUMER_GROUP procedure are not specified, then they remain unchanged in the data dictionary.

DELETE_CONSUMER_GROUP Procedure

This procedure lets you delete entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (
   consumer_group IN VARCHAR2);
```

Parameters

Table 58–8 DELETE_CONSUMER_GROUP Procedure Parameters

Parameters	Description
consumer_group	Name of consumer group to be deleted.

CREATE_PLAN_DIRECTIVE Procedure

This procedure lets you create resource plan directives.For release 8.2 new_max_ active_sess_target_mth was renamed new_active_sess_pool_mth and several new parameters added.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   plan                      IN VARCHAR2,
   group_or_subplan          IN VARCHAR2,
   comment                   IN VARCHAR2,
   cpu_p1                    IN NUMBER   DEFAULT NULL,
   cpu_p2                    IN NUMBER   DEFAULT NULL,
   cpu_p3                    IN NUMBER   DEFAULT NULL,
   cpu_p4                    IN NUMBER   DEFAULT NULL,
   cpu_p5                    IN NUMBER   DEFAULT NULL,
   cpu_p6                    IN NUMBER   DEFAULT NULL,
   cpu_p7                    IN NUMBER   DEFAULT NULL,
   cpu_p8                    IN NUMBER   DEFAULT NULL,
   active_sess_pool_p1       IN NUMBER   DEFAULT UNLIMITED,
   queueing_p1               IN NUMBER   DEFAULT UNLIMITED,
   switch_group              IN VARCHAR2 DEFAULT NULL,
   switch_time               IN NUMBER   DEFAULT UNLIMITED,
   switch_estimate           IN BOOLEAN  DEFAULT FALSE,
   max_est_exec_time         IN NUMBER   DEFAULT UNLIMITED,
   undo_pool                 IN NUMBER   DEFAULT UNLIMITED,
   parallel_degree_limit_p1  IN NUMBER   DEFAULT UNLIMITED);
```

Parameters

Table 58–9 CREATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of consumer group or subplan.
comment	Comment for the plan directive.
cpu_p1	First parameter for the CPU resource allocation method.
cpu_p2	Second parameter for the CPU resource allocation method.
cpu_p3	Third parameter for the CPU resource allocation method.

Table 58–9 CREATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
cpu_p4	Fourth parameter for the CPU resource allocation method.
cpu_p5	Fifth parameter for the CPU resource allocation method.
cpu_p6	Sixth parameter for the CPU resource allocation method.
cpu_p7	Seventh parameter for the CPU resource allocation method.
cpu_p8	Eighth parameter for the CPU resource allocation method.
active_sess_pool_p1	First parameter for the maximum active sessions allocation method (Reserved for future use).
queueing_p1	queue timeout in seconds
switch_group	group to switch into once switch time is reached
switch_time	switch time
switch_estimate	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.
max_est_exec_time	maximum estimated execution time in seconds
undo_pool	undo pool size for the consumer group, in Kbytes
parallel_degree_limit_p1	First parameter for the degree of parallelism allocation method.

All parameters default to NULL. However, for the EMPHASIS CPU resource allocation method, this case would starve all the users.

UPDATE_PLAN_DIRECTIVE Procedure

This procedure lets you update resource plan directives. For release 8.2 new_max_active_sess_target_mth was renamed new_active_sess_pool_mth and several new parameters added

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (
   plan                          IN VARCHAR2,
   group_or_subplan              IN VARCHAR2,
   new_comment                   IN VARCHAR2 DEFAULT NULL,
   new_cpu_p1                    IN NUMBER   DEFAULT NULL,
   new_cpu_p2                    IN NUMBER   DEFAULT NULL,
   new_cpu_p3                    IN NUMBER   DEFAULT NULL,
   new_cpu_p4                    IN NUMBER   DEFAULT NULL,
   new_cpu_p5                    IN NUMBER   DEFAULT NULL,
   new_cpu_p6                    IN NUMBER   DEFAULT NULL,
   new_cpu_p7                    IN NUMBER   DEFAULT NULL,
   new_cpu_p8                    IN NUMBER   DEFAULT NULL,
   new_active_sess_pool_p1       IN NUMBER   DEFAULT NULL,
   new_queueing_p1               IN NUMBER   DEFAULT NULL,
   new_parallel_degree_limit_p1  IN NUMBER   DEFAULT NULL
   new_switch_group              IN VARCHAR2 DEFAULT NULL,
   new_switch_time               IN NUMBER   DEFAULT NULL,
   new_switch_estimate           IN BOOLEAN  DEFAULT FALSE,
   new_max_est_exec_time         IN NUMBER   DEFAULT NULL,
   new_undo_pool                 IN NUMBER   DEFAULT UNLIMITED);
```

Parameters

Table 58–10 UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of consumer group or subplan.
new_comment	Comment for the plan directive.
new_cpu_p1	First parameter for the CPU resource allocation method.
new_cpu_p2	Second parameter for the CPU resource allocation method.
new_cpu_p3	Third parameter for the CPU resource allocation method.

Table 58–10 UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
new_cpu_p4	Fourth parameter for the CPU resource allocation method.
new_cpu_p5	Fifth parameter for the CPU resource allocation method.
new_cpu_p6	Sixth parameter for the CPU resource allocation method.
new_cpu_p7	Seventh parameter for the CPU resource allocation method.
new_cpu_p8	Eighth parameter for the CPU resource allocation method.
new_active_sess_pool_p1	First parameter for the maximum active sessions allocation method (Reserved for future use).
new_queueing_p1	queue timeout in seconds
new_switch_group	group to switch into once switch time is reached
new_switch_time	switch time
new_switch_estimate	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.
new_max_est_exec_time	maximum estimated execution time in seconds
new_undo_pool	undo pool size for the consumer group, in Kbytes
new_parallel_degree_limit_p1	First parameter for the degree of parallelism allocation method.

If the parameters for UPDATE_PLAN_DIRECTIVE are left unspecified, then they remain unchanged in the data dictionary.

DELETE_PLAN_DIRECTIVE Procedure

This procedure lets you delete resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE (
   plan             IN VARCHAR2,
   group_or_subplan  IN VARCHAR2);
```

Parameters

Table 58–11 DELETE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of group or subplan.

CREATE_PENDING_AREA Procedure

This procedure lets you make changes to resource manager objects.

All changes to the plan schema must be done within a pending area. The pending area can be thought of as a "scratch" area for plan schema changes. The administrator creates this pending area, makes changes as necessary, possibly validates these changes, and only when the submit is completed do these changes become active.

You may, at any time while the pending area is active, view the current plan schema with your changes by selecting from the appropriate user views.

At any time, you may clear the pending area if you want to stop the current changes. You may also call the VALIDATE procedure to confirm whether the changes you has made are valid. You do not have to do your changes in a given order to maintain a consistent group of entries. These checks are also implicitly done when the pending area is submitted.

> **Note:** Oracle allows "orphan" consumer groups (in other words, consumer groups that have no plan directives that refer to them). This is in anticipation that an administrator may want to create a consumer group that is not currently being used, but will be used in the future.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

Usage Notes

The following rules must be adhered to, and they are checked whenever the validate or submit procedures are executed:

1. No plan schema may contain any loops.
2. All plans and consumer groups referred to by plan directives must exist.
3. All plans must have plan directives that refer to either plans or consumer groups.
4. All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
5. No plan may be deleted that is currently being used as a top plan by an active instance.
6. For Oracle8*i*, the plan directive parameter, `parallel_degree_limit_p1`, may only appear in plan directives that refer to consumer groups (that is, not at subplans).
7. There cannot be more than 32 plan directives coming from any given plan (that is, no plan can have more than 32 children).
8. There cannot be more than 32 consumer groups in any active plan schema.
9. Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
10. There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema.This ensures that a session not covered by the currently active plan is allocated resources as specified by the `OTHER_GROUPS` directive.

If any of the preceding rules are broken when checked by the `VALIDATE` or `SUBMIT` procedures, then an informative error message is returned. You may then make changes to fix the problem(s) and reissue the validate or submit procedures.

VALIDATE_PENDING_AREA Procedure

This procedure lets you validate pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

CLEAR_PENDING_AREA Procedure

This procedure lets you clear pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

SUBMIT_PENDING_AREA Procedure

This procedure lets you submit pending changes for the resource manager: It clears the pending area after validating and committing the changes (if valid).

> **Note:** A call to SUBMIT_PENDING_AREA may fail even if VALIDATE_PENDING_AREA succeeds. This may happen if a plan being deleted is loaded by an instance after a call to VALIDATE_PENDING_AREA, but before a call to SUBMIT_PENDING_AREA.

Syntax

```
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

Example

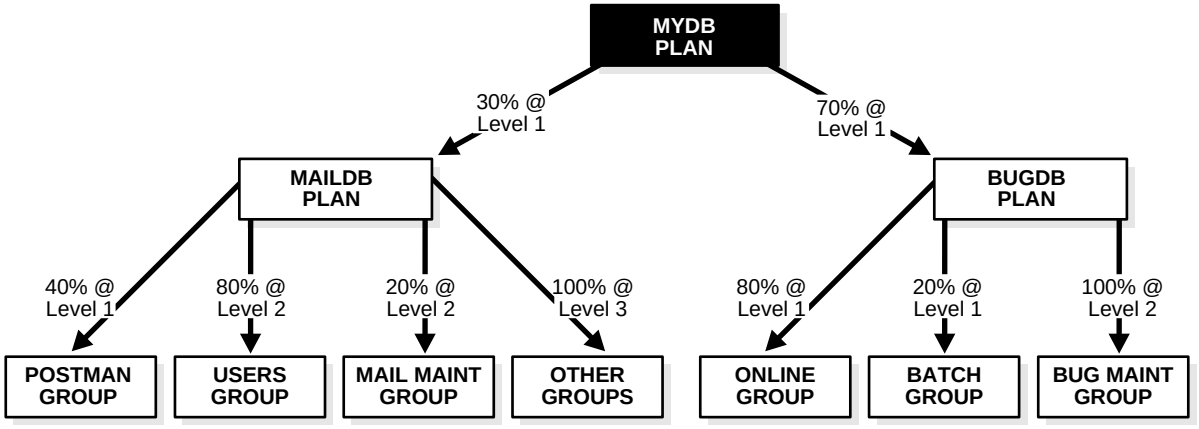
One of the advantages of plans is that they can refer to each other. The entries in a plan can either be consumer groups or subplans. For example, the following is also a set of valid CPU plan directives:

Table 58–12 MYDB PLAN CPU Plan Directives

Subplan/Group	CPU_Level 1
MAILDB Plan	30%
BUGDB Plan	70%

If these plan directives were in effect and there were an infinite number of runnable sessions in all consumer groups, then the MAILDB plan would be assigned 30% of the available CPU resources, while the BUGDB plan would be assigned 70% of the available CPU resources. Breaking this further down, sessions in the "Postman" consumer group would be run 12% (40% of 30%) of the time, while sessions in the "Online" consumer group would be run 56% (80% of 70%) of the time. Figure 58–1 diagram depicts this scenario:

Figure 58–1 Resource Manager Scenario



Conceptually below the consumer groups are the active sessions. In other words, a session belongs to a resource consumer group, and this consumer group is used by a plan to determine allocation of processing resources.

A multiplan (plan with one or more subplans) definition of CPU plan directives cannot be collapsed into a single plan with one set of plan directives, because each plan is its own entity. The CPU quanta that is allotted to a plan or subplan gets used only within that plan, unless that plan contains no consumer groups with active sessions. Therefore, in this example, if the Bug Maintenance Group did not use any of its quanta, then it would get recycled within that plan, thus going back to level 1 within the BUGDB PLAN. If the multiplan definition in the preceding example got collapsed into a single plan with multiple consumer groups, then there would be no way to explicitly recycle the Bug Maintenance Group's unused quanta. It would have to be recycled globally, thus giving the mail sessions an opportunity to use it.

The resources for a database can be partitioned at a high level among multiple applications and then repartitioned within an application. If a given group within an application does not need all the resources it is assigned, then the resource is only repartitioned within the same application.

The following example uses the default plan and consumer group allocation methods:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
```

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Batch_group',
COMMENT => 'Resource consumer group/method for bug users sessions who run batch jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maintenance_group',
   COMMENT => 'Resource consumer group/method for users sessions who maintain
   the bug  db');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Postman_group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maintenance_group',
   COMMENT => 'Resource consumer group/method for users sessions who maintain the mail
   db');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN =>
'Bug_Online_group',
   COMMENT => 'online bug users sessions at level 1', CPU_P1 => 80, CPU_P2=> 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN =>
'Bug_Batch_group',
   COMMENT => 'batch bug users sessions at level 1', CPU_P1 => 20, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN =>
'Bug_Maintenance_group',
   COMMENT => 'bug maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 100,
PARALLEL_DEGREE_LIMIT_P1 => 3);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan', GROUP_OR_SUBPLAN =>
'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0, CPU_P3 =>
100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_SUBPLAN =>
'Mail_Postman_group',
   COMMENT => 'mail postman at level 1', CPU_P1 => 40, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_SUBPLAN =>
'Mail_users_group',
   COMMENT => 'mail users sessions at level 2', CPU_P1 => 0, CPU_P2 => 80,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_SUBPLAN =>
'Mail_Maintenance_group',
   COMMENT => 'mail maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 20,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan', GROUP_OR_SUBPLAN =>
'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0, CPU_P3 =>
```

```
100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan', GROUP_OR_SUBPLAN =>
'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', CPU_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan', GROUP_OR_SUBPLAN =>
'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', CPU_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
end;
```

The preceding call to VALIDATE_PENDING_AREA is optional, because the validation is implicitly done in SUBMIT_PENDING_AREA.

SET_INITIAL_CONSUMER_GROUP Procedure

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. This procedure sets the initial resource consumer group for a user.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
   user           IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 58–13 SET_INITIAL_CONSUMER_GROUP Procedure Parameters

Parameters	Description
user	Name of the user.
consumer_group	The user's initial consumer group.

Usage Notes

The ADMINISTER_RESOURCE_MANAGER or the ALTER USER system privilege are required to be able to execute this procedure. The user, or PUBLIC, must be directly granted switch privilege to a consumer group before it can be set to be the user's initial consumer group. Switch privilege for the initial consumer group cannot come from a role granted to that user.

Note: These semantics are similar to those for ALTER USER DEFAULT ROLE.

If the initial consumer group for a user has never been set, then the user's initial consumer group is automatically the consumer group: DEFAULT_CONSUMER_GROUP.

DEFAULT_CONSUMER_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group now have DEFAULT_CONSUMER_GROUP as their initial consumer group. All currently active sessions belonging to a deleted consumer group are switched to DEFAULT_CONSUMER_GROUP.

SWITCH_CONSUMER_GROUP_FOR_SESS Procedure

This procedure lets you change the resource consumer group of a specific session. It also changes the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (
   session_id      IN NUMBER,
   session_serial  IN NUMBER,
   consumer_group  IN VARCHAR2);
```

Parameters

Table 58–14 SWITCH_CONSUMER_GROUP_FOR_SESS Procedure Parameters

Parameter	Description
session_id	SID column from the view V$SESSION.
session_serial	SERIAL# column from view V$SESSION.
consumer_group	Name of the consumer group to switch to.

SWITCH_CONSUMER_GROUP_FOR_USER Procedure

This procedure lets you change the resource consumer group for all sessions with a given user ID. It also change the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (
   user            IN VARCHAR2,
   consumer_group  IN VARCHAR2);
```

Parameters

Table 58–15 SWITCH_CONSUMER_GROUP_FOR_USER Procedure Parameters

Parameter	Description
user	Name of the user.
consumer_group	Name of the consumer group to switch to.

Usage Notes

The SWITCH_CONSUMER_GROUP_FOR_SESS and SWITCH_CONSUMER_GROUP_FOR_USER procedures let you to raise or lower the allocation of CPU resources of certain sessions or users. This provides a functionality similar to the nice command on UNIX.

These procedures cause the session to be moved into the newly specified consumer group immediately.

59

DBMS_RESOURCE_MANAGER_PRIVS

The DBMS_RESOURCE_MANAGER_PRIVS package maintains privileges associated with the Resource Manager.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle9i Database Administrator's Guide.*

This chapter discusses the following topics:

Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms

Table 59–1 DBMS_RESOURCE_MANAGER_PRIVS Subprograms

Subprogram	Description
GRANT_SYSTEM_PRIVILEGE Procedure on page 59-2	Performs a grant of a system privilege.
REVOKE_SYSTEM_PRIVILEGE Procedure on page 59-3	Performs a revoke of a system privilege.
GRANT_SWITCH_CONSUMER_GROUP Procedure on page 59-3	Grants the privilege to switch to resource consumer groups.
REVOKE_SWITCH_CONSUMER_GROUP Procedure on page 59-5	Revokes the privilege to switch to resource consumer groups.

GRANT_SYSTEM_PRIVILEGE Procedure

This procedure performs a grant of a system privilege to a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
   grantee_name   IN VARCHAR2,
   privilege_name  IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER',
   admin_option   IN BOOLEAN);
```

Parameters

Table 59–2 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
privilege_name	Name of the privilege to be granted.
admin_option	TRUE if the grant is with admin_option, FALSE otherwise.

Currently, Oracle provides only one system privilege for the Resource Manager: ADMINISTER_RESOURCE_MANAGER. Database administrators have this system privilege with the admin option. The grantee and the revokee can either be a user or

a role. Users that have been granted the system privilege with the admin option can also grant this privilege to others.

Example

The following call grants this privilege to a user called scott without the admin option:

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
   grantee_name => 'scott',
   admin_option => FALSE);
```

REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure performs a revoke of a system privilege from a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (
   revokee_name   IN VARCHAR2,
   privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');
```

Parameters

Table 59–3 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
revokee_name	Name of the user or role from whom privilege is to be revoked.
privilege_name	Name of the privilege to be revoked.

Example

The following call revokes the ADMINISTER_RESOURCE_MANAGER from user scott:

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE ('scott');
```

GRANT_SWITCH_CONSUMER_GROUP Procedure

This procedure grants the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   grantee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2,
   grant_option   IN BOOLEAN);
```

Parameters

Table 59–4 GRANT_SWITCH_CONSUMER_GROUP Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
consumer_group	Name of consumer group.
grant_option	TRUE if grantee should be allowed to grant access, FALSE otherwise.

Usage Notes

If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that consumer group.

If the grant_option parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others.

In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user.

See Also: Chapter 58, "DBMS_RESOURCE_MANAGER"

Example

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_COMSUMER_GROUP (
   'scott', 'mail_maintenance_group', true);
```

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
   'scott', 'mail_maintenance_group');
```

REVOKE_SWITCH_CONSUMER_GROUP Procedure

This procedure revokes the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   revokee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 59–5 REVOKE_SWITCH_CONSUMER_GROUP Procedure Parameter

Parameter	Description
revokee_name	Name of user/role from which to revoke access.
consumer_group	Name of consumer group.

Usage Notes

If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.

If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP consumer group when logging in.

If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group through that role will not be able to switch to that consumer group.

If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group through PUBLIC will not be able to switch to that consumer group.

Example

The following example revokes the privileges to switch to mail_maintenance_group from Scott:

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   'scott', 'mail_maintenance_group');
```

60

DBMS_RESUMABLE

With DBMS_RESUMABLE, you can suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution. Thus, you can write applications without worrying about running into space-related errors.

When you suspend a statement, you should log the suspension in the alert log. You should also register a procedure to be executed when the statement is suspended. Using a view, you can monitor the progress of the statement and indicate whether the statement is currently executing or suspended.

Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held during a statement suspend and resume. When the error condition disappears, the suspended statement automatically resumes execution. A resumable space allocation can be suspended and resumed multiple times during execution.

A suspension timeout interval is associated with resumable space allocations. A resumable space allocation that is suspended for the timeout interval (the default is two hours) wakes up and returns an exception to the user. A suspended statement may be forced to throw an exception using the DMBS_RESUMABLE.ABORT( ) procedure.

This chapter discusses the following topics:

Summary of DBMS_RESUMABLE Subprograms

Table 60–1 DBMS_RESUMABLE Subprograms

Subprogram	Description
ABORT Procedure on page 60-2	Aborts a suspended resumable space allocation.
GET_SESSION_TIMEOUT Function on page 60-3	Returns the current timeout value of the resumable space allocations for a session with `session_id`.
SET_SESSION_TIMEOUT Procedure on page 60-3	Sets the timeout of resumable space allocations for a session with `session_id`.
GET_TIMEOUT Function on page 60-4	Returns the current timeout value of resumable space allocations for the current session.
SET_TIMEOUT Procedure on page 60-4	Sets the timeout of resumable space allocations for the current session.
SPACE_ERROR_INFO Function on page 60-5	Looks for space-related errors in the error stack. If it cannot find a space-related error, it will return `FALSE`.

ABORT Procedure

This procedure aborts a suspended resumable space allocation. The parameter `session_id` is the session ID in which the statement is executed. For a parallel DML/DDL, `session_id` is any session ID that participates in the parallel DML/DDL. This operation is guaranteed to succeed. The procedure can be called either inside or outside of the `AFTER SUSPEND` trigger.

To call an `ABORT` procedure, you must be the owner of the session with `session_id`, have `ALTER SYSTEM` privileges, or be a DBA.

Syntax

```
DBMS_RESUMABLE.ABORT (
   session_id  IN NUMBER);
```

Parameters

Table 60–2 ABORT Procedure Parameters

Parameter	Description
session_id	The session identifier of the resumable space allocation.

GET_SESSION_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for a session with session_id. The timeout is returned in seconds. If session_id does not exist, the GET_SESSION_TIMEOUT function returns -1.

Syntax

```
DBMS_RESUMABLE.GET_SESSION_TIMEOUT (
   session_id  IN NUMBER);
```

Parameters

Table 60–3 GET_SESSION_TIMEOUT Function Parameters

Parameter	Description
session_id	The session identifier of the resumable space allocation.

SET_SESSION_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for a session with session_id. The timeout is returned in seconds. The new timeout setting applies to the session immediately. If session_id does not exist, no operation occurs.

Syntax

```
DBMS_RESUMABLE.SET_SESSION_TIMEOUT (
   session_id  IN NUMBER,
   timeout     IN NUMBER);
```

Parameters

Table 60–4 SET_SESSION_TIMEOUT Procedure Parameters

Parameter	Description
session_id	The session identifier of the resumable space allocation.
timeout	The timeout of the resumable space allocation.

GET_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for the current session. The returned value is in seconds. If session_id does not exist, the GET_TIMEOUT function returns -1.

Syntax

```
DBMS_RESUMABLE.GET_TIMEOUT;
```

SET_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for the current session. The timeout is returned in seconds. The new timeout setting applies to the session immediately. If session_id does not exist, no operation occurs.

Syntax

```
DBMS_RESUMABLE.SET_TIMEOUT (
   timeout  IN NUMBER);
```

Parameters

Table 60–5 SET_TIMEOUT Procedure Parameters

Parameter	Description
timeout	The timeout of the resumable space allocation.

SPACE_ERROR_INFO Function

This function looks for space-related errors in the error stack. If it cannot find a space related error, it will return FALSE. Otherwise, TRUE is returned and information about the particular object that causes the space error is returned.

Syntax

```
DBMS_RESUMABLE.SPACE_ERROR_INFO
   error_type        OUT VARCHAR2,
   object_type       OUT VARCHAR2,
   object_owner      OUT VARCHAR2,
   table_space_name  OUT VARCHAR2,
   object_name       OUT VARCHAR2,
   sub_object_name   OUT VARCHAR2)
return boolean;
```

Parameters

Table 60–6 SPACE_ERROR_INFO Function Parameters

Parameter	Description
error_type	The space error type. It will be one of the following: ■ NO MORE SPACE ■ MAX EXTENTS REACHED ■ SPACE QUOTA EXCEEDED

Table 60–6 SPACE_ERROR_INFO Function Parameters

Parameter	Description
object_type	The object type. It will be one of the following: ■ TABLE SPACE ■ ROLLBACK SEGMENT ■ UNDO SEGMENT ■ TABLE ■ INDEX ■ CLUSTER ■ TEMP SEGMENT ■ INDEX PARTITION ■ TABLE PARTITION ■ LOB SEGMENT ■ TABLE SUBPARTITION ■ INDEX SUBPARTITION ■ LOB SUBPARTITION
object_owner	The owner of the object. NULL if it cannot be determined.
table_space_name	The table space where the object resides. NULL if it cannot be determined.
object_name	The name of rollback segment, temp segment, table, index, or cluster.
sub_object_name	The partition name or sub-partition name of LOB, TABLE, or INDEX. NULL if it cannot be determined.

61

DBMS_RLS

The DBMS_RLS package contains the fine-grained access control administrative interface. DBMS_RLS is available with the Enterprise Edition only.

> **See Also:** *Oracle9i Application Developer's Guide - Fundamentals* for a detailed example and more usage information on DBMS_RLS.

This chapter discusses the following topics:

Dynamic Predicates

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
   'hr', 'employees', 'emp_policy', 'hr', 'emp_sec', 'select');
```

Whenever the EMPLOYEES table, under the HR schema, is referenced in a query or subquery (SELECT), the server calls the EMP_SEC function (under the HR schema). This returns a predicate specific to the current user for the EMP_POLICY policy. The policy function may generate the predicates based on the session environment variables available during the function call. These variables usually appear in the form of application contexts.

The server then produces a transient view with the text:

```
SELECT * FROM hr.employees WHERE P1
```

Here, P1 (for example, where SAL > 10000, or even a subquery) is the predicate returned from the EMP_SEC function. The server treats the EMPLOYEES table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy-protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users do not require EXECUTE privilege on the policy function, because the server makes the call with the function definer's right.

Note: The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; that is, no JOIN, ORDER BY, GROUP BY, and so on.

DBMS_RLS also provides the interface to drop, enable, and disable security policies. For example, you can drop or disable the EMP_POLICY with the following PL/SQL statements:

```
DBMS_RLS.DROP_POLICY('hr', 'employees', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('hr', 'employees', 'emp_policy', FALSE)
```

Security

A security check is performed when the transient view is created with a subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for security check and object lookup.

Usage Notes

The DBMS_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Views of current cursors and corresponding predicates are available from v$vpd_policies.

A synonym can reference only a view or a table.

Summary of DBMS_RLS Subprograms

Table 61–1 DBMS_RLS Subprograms

Subprogram	Description
ADD_POLICY Procedure on page 61-4	Adds a fine-grained access control policy to a table, view, or synonym..
DROP_POLICY Procedure on page 61-7	Drops a fine-grained access control policy from a table, view, or synonym..

Table 61–1 DBMS_RLS Subprograms

Subprogram	Description
REFRESH_POLICY Procedure on page 61-7	Causes all the cached statements associated with the policy to be reparsed.
ENABLE_POLICY Procedure on page 61-8	Enables or disables a fine-grained access control policy.
CREATE_POLICY_GROUP Procedure on page 61-9	Creates a policy group.
ADD_GROUPED_POLICY Procedure on page 61-10	Adds a policy associated with a policy group.
ADD_POLICY_CONTEXT Procedure on page 61-11	Adds the context for the active application.
DELETE_POLICY_GROUP Procedure on page 61-13	Deletes a policy group.
DROP_GROUPED_POLICY Procedure on page 61-13	Drops a policy associated with a policy group.
DROP_POLICY_CONTEXT Procedure on page 61-14	Drops a driving context from the object so that it will have one less driving context.
ENABLE__GROUPED_POLICY Procedure on page 61-15	Enables or disables a row-level group security policy.
REFRESH_GROUPED_POLICY Procedure on page 61-15	Reparses the SQL statements associated with a refreshed policy.

ADD_POLICY Procedure

This procedure adds a fine-grained access control policy to a table , view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: "Usage Notes" on page 61-3

A `COMMIT` is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ADD_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   policy_name     IN VARCHAR2,
   function_schema IN VARCHAR2 NULL,
   policy_function IN VARCHAR2,
   statement_types IN VARCHAR2 NULL,
   update_check    IN BOOLEAN  FALSE,
   enable          IN BOOLEAN  TRUE,
   static_policy   IN BOOLEAN  FALSE);
```

Parameters

Table 61–2 ADD_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table,view, or synonym (current default schema, if NULL).
object_name	Name of table, view, or synonym to which the policy is added.
policy_name	Name of policy to be added. It must be unique for the same table or view.
function_schema	Schema of the policy function (current default schema, if NULL).
policy_function	Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of SELECT, INSERT, UPDATE, and DELETE. The default is to apply to all of these types.
update_check	Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting update_check to TRUE causes the server to also check the policy against the value after insert or update.
enable	Indicates if the policy is enabled when it is added. The default is TRUE
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.

Usage Notes

- The server invokes the policy function once for each cursor and will therefore improve performance for statement parsing and execution. Declaring a policy function DETERMINISTIC does not affect performance.
- SYS is free of any security policy.
- The policy functions which generate dynamic predicates are called by the server. Following is the interface for the function:

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2
--- object_schema is the schema owning the table of view.
--- object_name is the name of table, view, or synonym to which the policy
applies.
```

  The maximum length of the predicate that the policy function can return is 32K.

- The policy functions must have the purity level of WNDS (write no database state).

  See Also: The *Oracle9i Application Developer's Guide - Fundamentals* has more details about the RESTRICT_REFERENCES pragma.

- Dynamic predicates generated out of different policies for the same object have the combined effect of a conjunction (ANDed) of all the predicates.
- The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.
- If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.
- When a table alias is required (for example, parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like "select c1, c2, ... from tab where <predicate>".
- The checking of the validity of the function is done at runtime for ease of installation and other dependency issues during import/export.

DROP_POLICY Procedure

This procedure drops a fine-grained access control policy from a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: "Usage Notes" on page 61-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.DROP_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2);
```

Parameters

Table 61–3 DROP_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view or synonym (current default schema if `NULL`).
`object_name`	Name of table, view, or synonym.
`policy_name`	Name of policy to be dropped from table, view, or synonym..

REFRESH_POLICY Procedure

This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: "Usage Notes" on page 61-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.REFRESH_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2 NULL,
   policy_name   IN VARCHAR2 NULL);
```

Parameters

Table 61–4 REFRESH_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym.
`object_name`	Name of table, view, or synonym with which the policy is associated.
`policy_name`	Name of policy to be refreshed.

Errors

The procedure returns an error if it tries to refresh a disabled policy.

ENABLE_POLICY Procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: "Usage Notes" on page 61-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ENABLE_POLICY (
```

```
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2,
   enable        IN BOOLEAN);
```

Parameters

Table 61–5 ENABLE_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing table, view, or synonym (current default schema if NULL).
object_name	Name of table, view, or synonym with which the policy is associated.
policy_name	Name of policy to be enabled or disabled.
enable	TRUE to enable the policy, FALSE to disable the policy.

CREATE_POLICY_GROUP Procedure

This procedure creates a policy group.

Syntax

```
DBMS_RLS.CREATE_POLICY_GROUP (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2 );
```

Parameters

Table 61–6 CREATE_POLICY_GROUP Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym.
object_name	Name of the table, view, or synonym to which the policy is added.
policy_group	Name of the policy group that the policy belongs to.

Usage Notes

The group must be unique for each table or view.

ADD_GROUPED_POLICY Procedure

This procedure adds a policy associated with a policy group.

Syntax

```
DBMS_RLS.ADD_GROUPED_POLICY(
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2,
   policy_name     VARCHAR2,
   function_schema VARCHAR2,
   policy_function VARCHAR2,
   statement_types VARCHAR2,
   update_check    BOOLEAN,
   enabled         BOOLEAN,
   static_policy   BOOLEAN  FALSE );
```

Parameters

Table 61–7 ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to.
policy_name	The name of the policy; must be unique for the same table or view.
function_schema	The schema owning the policy function.
policy_function	The name of the function that generates a predicate for the policy. If the function is defined within a package, the name of the package must be present.
statement_types	The list of statement types to which the policy can apply. It can be any combination of SELECT, INSERT, UPDATE, or DELETE. Optional.

Table 61–7 ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
update_check	For INSERT and UPDATE statements only, setting update_check to TRUE causes the server to check the policy against the value after INSERT or UPDATE.
enable	Indicates if the policy is enable when it is added. The default is TRUE.
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.

Usage Notes

- The server invokes the policy function once for each cursor and will therefore improve performance for statement parsing and execution. Declaring a policy function DETERMINISTIC does not affect performance.
- This procedure adds a policy to the specified table, view, or synonym and associates the policy with the specified policy group.
- The policy group must have been created using the CREATE_POLICY_GROUP interface.
- The policy name must be unique within a policy group for a specific object.
- Policies from the default policy group, SYS_DEFAULT, are always executed regardless of the active policy group; however, fine-grained access control policies do not apply to users with EXEMPT ACCESS POLICY system privilege.

ADD_POLICY_CONTEXT Procedure

This procedure adds the context for the active application.

Syntax

```
DBMS_RLS.ADD_POLICY_CONTEXT (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   namespace       VARCHAR2,
   attribute       VARCHAR2 );
```

Parameters

Table 61–8 ADD_POLICY_CONTEXT Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
namespace	The namespace of the driving context
attribute	The attribute of the driving context.

Usage Notes

Note the following:

- This procedure indicates the application context that drives the enforcement of policies; this is the context that determines which application is running.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is NULL, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the SYS_DEFAULT policy group.
- To add a policy to table hr.employees in group access_control_group, the following command is issued:

```
DBMS_RLS.ADD_GROUPED_POLICY('hr','employees','access_control_
group','policy1','SYS', 'HR.ACCESS');
```

DELETE_POLICY_GROUP Procedure

This procedure deletes a policy group.

Syntax

```
DBMS_RLS.DELETE_POLICY_GROUP (
  object_schema   VARCHAR2,
  object_name     VARCHAR2,
  policy_group    VARCHAR2 );
```

Parameters

Table 61–9 DELETE_POLICY_GROUP Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to

Usage Notes

Note the following:

- This procedure deletes a policy group for the specified table, view, or synonym.
- No policy can be in the policy group.

DROP_GROUPED_POLICY Procedure

This procedure drops a policy associated with a policy group.

Syntax

```
DBMS_RLS.DROP_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   policy_group    VARCHAR2,
   policy_name     VARCHAR2 );
```

Parameters

Table 61–10 DROP_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is dropped.
policy_group	The name of the policy group that the policy belongs to.
policy_name	The name of the policy.

DROP_POLICY_CONTEXT Procedure

This procedure drops a driving context from the object so that it will have one less driving context.

Syntax

```
DBMS_RLS.DROP_POLICY_CONTEXT (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   namespace       VARCHAR2,
   attribute       VARCHAR2 );
```

Parameters

Table 61–11 DROP_POLICY_CONTEXT Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym..
object_name	The name of the table, view, or synonym to which the policy is dropped.
namespace	The namespace of the driving context.
attribute	The attribute of the driving context.

ENABLE__GROUPED_POLICY Procedure

This procedure enables or disables a row-level group security policy.

Syntax

```
DBMS_RLS.ENABLE_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   group_name      VARCHAR2,
   policy_name     VARCHAR2,
   enable          BOOLEAN  );
```

Parameters

Table 61–12 ENABLE_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym with which the policy is associated.
group_name	The name of the group of the policy.
policy_name	The name of the policy to be enabled or disabled.
enable	TRUE enables the policy; FALSE disables the policy.

Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is enabled when it is created.

REFRESH_GROUPED_POLICY Procedure

This procedure reparses the SQL statements associated with a refreshed policy.

Syntax

```
DBMS_RLS.REFRESH_GROUPED_POLICY (
   object_schema   VARCHAR2,
   object_name     VARCHAR2,
   group_name      VARCHAR2,
   policy_name     VARCHAR2 );
```

Parameters

Table 61–13 REFRESH_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema containing the table, view, or synonym..
object_name	The name of the table, view, or synonym with which the policy is associated.
group_name	The name of the group of the policy.
policy_name	The name of the policy.

Usage Notes

- This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to the policy has immediate effect after the procedure is executed.
- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.

62

DBMS_ROWID

The DBMS_ROWID package lets you create ROWIDs and obtain information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other ROWID components without writing code to interpret the base-64 character external ROWID.

Note: DBMS_ROWID is not to be used with universal ROWIDs (UROWIDs).

This chapter discusses the following topics:

Usage Notes

Some of the functions in this package take a single parameter, such as a ROWID. This can be a character or a PL/SLQ ROWID, either restricted or extended, as required.

You can call the DBMS_ROWID functions and procedures from PL/SQL code, and you can also use the functions in SQL statements.

Note: ROWID_INFO is a procedure. It can only be used in PL/SQL code.

You can use functions from the DBMS_ROWID package just like built-in SQL functions; in other words, you can use them wherever you can use an expression. In this example, the ROWID_BLOCK_NUMBER function is used to return just the block number of a single row in the EMP table:

```
SELECT dbms_rowid.rowid_block_number(rowid)
   FROM emp
   WHERE ename = 'KING';
```

Troubleshooting Use of the RESTRICT_REFERENCES Pragma

If Oracle returns the error "ORA:452, 0, 'Subprogram '%s' violates its associated pragma' for pragma restrict_references", it could mean the violation is due to:

- A problem with the current procedure or function
- Calling a procedure or function without a pragma or due to calling one with a less restrictive pragma
- Calling a package procedure or function that touches the initialization code in a package or that sets the default values

PL/SQL Example

This example returns the ROWID for a row in the EMP table, extracts the data object number from the ROWID, using the ROWID_OBJECT function in the DBMS_ROWID package, then displays the object number:

```
DECLARE
  object_no   INTEGER;
  row_id      ROWID;
  ...
BEGIN
  SELECT ROWID INTO row_id FROM emp
```

```
  WHERE empno = 7499;
object_no := dbms_rowid.rowid_object(row_id);
dbms_output.put_line('The obj. # is '|| object_no);
...
```

Requirements

This package runs with the privileges of calling user, rather than the package owner ('sys').

ROWID Types

The types are as follows:

- RESTRICTED—restricted ROWID
- EXTENDED—extended ROWID

For example:

```
rowid_type_restricted constant integer := 0;
rowid_type_extended   constant integer := 1;
```

Note: Extended ROWIDs are only used in Oracle8*i* and higher.

ROWID Verification Results

Result	Description
VALID	Valid ROWID
INVALID	Invalid ROWID

For example:

```
rowid_is_valid   constant integer := 0;
rowid_is_invalid constant integer := 1;
```

Object Types

Result	Description
UNDEFINED	Object Number not defined (for restricted ROWIDs)

For example:

```
rowid_object_undefined constant integer := 0;
```

ROWID Conversion Types

Result	Description
INTERNAL	Convert to/from column of ROWID type
EXTERNAL	Convert to/from string format

For example:

```
rowid_convert_internal constant integer := 0;
rowid_convert_external constant integer := 1;
```

Exceptions

Exception	Description
ROWID_INVALID	Invalid rowid format
ROWID_BAD_BLOCK	Block is beyond end of file

For example:

```
ROWID_INVALID exception;
   pragma exception_init(ROWID_INVALID, -1410);

ROWID_BAD_BLOCK exception;
   pragma exception_init(ROWID_BAD_BLOCK, -28516);
```

Summary of DBMS_ROWID Subprograms

Table 62–1 DBMS_ROWID Subprograms

Subprogram	Description
ROWID_CREATE Function on page 62-5	Creates a ROWID, for testing only.
ROWID_INFO Procedure on page 62-7	Returns the type and components of a ROWID.
ROWID_TYPE Function on page 62-8	Returns the ROWID type: 0 is restricted, 1 is extended.
ROWID_OBJECT Function on page 62-8	Returns the object number of the extended ROWID.
ROWID_RELATIVE_FNO Function on page 62-9	Returns the file number of a ROWID.
ROWID_BLOCK_NUMBER Function on page 62-10	Returns the block number of a ROWID.
ROWID_ROW_NUMBER Function on page 62-11	Returns the row number.
ROWID_TO_ABSOLUTE_FNO Function on page 62-11	Returns the absolute file number associated with the ROWID for a row in a specific table.
ROWID_TO_EXTENDED Function on page 62-13	Converts a ROWID from restricted format to extended.
ROWID_TO_RESTRICTED Function on page 62-14	Converts an extended ROWID to restricted format.
ROWID_VERIFY Function on page 62-15	Checks if a ROWID can be correctly extended by the ROWID_TO_EXTENDED function.

ROWID_CREATE Function

This function lets you create a ROWID, given the component parts as parameters.

This is useful for testing ROWID operations, because only the Oracle Server can create a valid ROWID that points to data in a database.

Syntax

```
DBMS_ROWID.ROWID_CREATE (
   rowid_type    IN NUMBER,
   object_number IN NUMBER,
   relative_fno  IN NUMBER,
```

```
  block_number  IN NUMBER,
  row_number    IN NUMBER)
 RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–2 ROWID_CREATE Function Parameters

Parameter	Description
rowid_type	Type (restricted or extended). Set the rowid_type parameter to 0 for a restricted ROWID. Set it to 1 to create an extended ROWID. If you specify rowid_type as 0, then the required object_number parameter is ignored, and ROWID_CREATE returns a restricted ROWID.
object_number	Data object number (rowid_object_undefined for restricted).
relative_fno	Relative file number.
block_number	Block number in this file.
file_number	File number in this block.

Example

Create a dummy extended ROWID:

```
my_rowid := DBMS_ROWID.ROWID_CREATE(1, 9999, 12, 1000, 13);
```

Find out what the rowid_object function returns:

```
obj_number := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

The variable obj_number now contains 9999.

ROWID_INFO Procedure

This procedure returns information about a ROWID, including its type (restricted or extended), and the components of the ROWID. This is a procedure, and it cannot be used in a SQL statement.

Syntax

```
DBMS_ROWID.ROWID_INFO (
   rowid_in         IN   ROWID,
   rowid_type       OUT  NUMBER,
   object_number    OUT  NUMBER,
   relative_fno     OUT  NUMBER,
   block_number     OUT  NUMBER,
   row_number       OUT  NUMBER);
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_info,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–3 ROWID_INFO Procedure Parameters

Parameter	Description
rowid_in	ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.
rowid_type	Returns type (restricted / extended).
object_number	Returns data object number (rowid_object_undefined for restricted).
relative_fno	Returns relative file number.
block_number	Returns block number in this file.
file_number	Returns file number in this block.

See Also: "ROWID_TYPE Function" on page 62-8

Example

This example reads back the values for the ROWID that you created in the ROWID_CREATE:

```
DBMS_ROWID.ROWID_INFO(my_rowid, rid_type, obj_num,
  file_num, block_num, row_num);

DBMS_OUTPUT.PUT_LINE('The type is ' || rid_type);
DBMS_OUTPUT.PUT_LINE('Data object number is ' || obj_num);
-- and so on...
```

ROWID_TYPE Function

This function returns 0 if the ROWID is a restricted ROWID, and 1 if it is extended.

Syntax

```
DBMS_ROWID.ROWID_TYPE (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_type,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–4 ROWID_TYPE Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

```
IF DBMS_ROWID.ROWID_TYPE(my_rowid) = 1 THEN
   my_obj_num := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

ROWID_OBJECT Function

This function returns the data object number for an extended ROWID. The function returns zero if the input ROWID is a restricted ROWID.

Syntax

```
DBMS_ROWID.ROWID_OBJECT (
```

```
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_object,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–5 ROWID_OBJECT Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

> **Note:** The ROWID_OBJECT_UNDEFINED constant is returned for restricted ROWIDs.

Example

```
SELECT dbms_rowid.rowid_object(ROWID)
   FROM emp
   WHERE empno = 7499;
```

ROWID_RELATIVE_FNO Function

This function returns the relative file number of the ROWID specified as the IN parameter. (The file number is relative to the tablespace.)

Syntax

```
DBMS_ROWID.ROWID_RELATIVE_FNO (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–6 ROWID_RELATIVE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

The example PL/SQL code fragment returns the relative file number:

```
DECLARE
   file_number    INTEGER;
   rowid_val      ROWID;
BEGIN
   SELECT ROWID INTO rowid_val
     FROM dept
     WHERE loc = 'Boston';
   file_number :=
     dbms_rowid.rowid_relative_fno(rowid_val);
    ...
```

ROWID_BLOCK_NUMBER Function

This function returns the database block number for the input ROWID.

Syntax

```
DBMS_ROWID.ROWID_BLOCK_NUMBER (
   row_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–7 ROWID_BLOCK_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

The example SQL statement selects the block number from a ROWID and inserts it into another table:

```
INSERT INTO T2 (SELECT dbms_rowid.rowid_block_number(ROWID)
   FROM some_table
   WHERE key_value = 42);
```

ROWID_ROW_NUMBER Function

This function extracts the row number from the ROWID IN parameter.

Syntax

```
DBMS_ROWID.ROWID_ROW_NUMBER (
   row_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_row_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–8 ROWID_ROW_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

Select a row number:

```
SELECT dbms_rowid.rowid_row_number(ROWID)
   FROM emp
   WHERE ename = 'ALLEN';
```

ROWID_TO_ABSOLUTE_FNO Function

This function extracts the absolute file number from a ROWID, where the file number is absolute for a row in a given schema and table. The schema name and the name

of the schema object (such as a table name) are provided as IN parameters for this function.

Syntax

```
DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO (
   row_id      IN ROWID,
   schema_name IN VARCHAR2,
   object_name IN VARCHAR2)
  RETURN NUMBER;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_to_absolute_fno,WNDS,WNPS,RNPS);
```

Parameters

Table 62–9 ROWID_TO_ABSOLUTE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.
schema_name	Name of the schema which contains the table.
object_name	Table name.

Example

```
DECLARE
   abs_fno        INTEGER;
   rowid_val      CHAR(18);
   object_name    VARCHAR2(20) := 'EMP';
BEGIN
    SELECT ROWID INTO rowid_val
     FROM emp
     WHERE empno = 9999;
     abs_fno := dbms_rowid.rowid_to_absolute_fno(
     rowid_val, 'SCOTT', object_name);
```

Note: For partitioned objects, the name must be a table name, not a partition or a sub/partition name.

ROWID_TO_EXTENDED Function

This function translates a restricted ROWID that addresses a row in a schema and table that you specify to the extended ROWID format. Later, it may be removed from this package into a different place.

Syntax

```
DBMS_ROWID.ROWID_TO_EXTENDED (
   old_rowid       IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER)
  RETURN ROWID;
```

Pragmas

```
   pragma RESTRICT_REFERENCES(rowid_to_extended,WNDS,WNPS,RNPS);
```

Parameters

Table 62–10 ROWID_TO_EXTENDED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
schema_name	Name of the schema which contains the table (optional).
object_name	Table name (optional).
conversion_type	rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type, or the character string).

Returns

ROWID_TO_EXTENDED returns the ROWID in the extended character format. If the input ROWID is NULL, then the function returns NULL. If a zero-valued ROWID is supplied (00000000.0000.0000), then a zero-valued restricted ROWID is returned.

Example

Assume that there is a table called RIDS in the schema SCOTT, and that the table contains a column ROWID_COL that holds ROWIDs (restricted), and a column

TABLE_COL that point to other tables in the SCOTT schema. You can convert the ROWIDs to extended format with the statement:

```
UPDATE SCOTT.RIDS
   SET rowid_col =
   dbms_rowid.rowid_to_extended (
      rowid_col, 'SCOTT", TABLE_COL, 0);
```

Usage Notes

If the schema and object names are provided as IN parameters, then this function verifies SELECT authority on the table named, and converts the restricted ROWID provided to an extended ROWID, using the data object number of the table. That ROWID_TO_EXTENDED returns a value, however, does not guarantee that the converted ROWID actually references a valid row in the table, either at the time that the function is called, or when the extended ROWID is actually used.

If the schema and object name are not provided (are passed as NULL), then this function attempts to fetch the page specified by the restricted ROWID provided. It treats the file number stored in this ROWID as the absolute file number. This can cause problems if the file has been dropped, and its number has been reused prior to the migration. If the fetched page belongs to a valid table, then the data object number of this table is used in converting to an extended ROWID value. This is very inefficient, and Oracle recommends doing this only as a last resort, when the target table is not known. The user must still know the correct table name at the time of using the converted value.

If an extended ROWID value is supplied, the data object number in the input extended ROWID is verified against the data object number computed from the table name parameter. If the two numbers do not match, the INVALID_ROWID exception is raised. If they do match, the input ROWID is returned.

> **See Also:** The ROWID_VERIFY Function has a method to determine if a given ROWID can be converted to the extended format.

ROWID_TO_RESTRICTED Function

This function converts an extended ROWID into restricted ROWID format.

Syntax

```
DBMS_ROWID.ROWID_TO_RESTRICTED (
```

```
  old_rowid       IN ROWID,
  conversion_type IN INTEGER)
 RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_restricted,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 62–11 ROWID_TO_RESTRICTED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
conversion_type	Internal or external - format of returned ROWID. rowid_convert_internal/external_convert_external (whether returned ROWID will be stored in a column of ROWID type or the character string)

ROWID_VERIFY Function

This function verifies the ROWID. It returns 0 if the input restricted ROWID can be converted to extended format, given the input schema name and table name, and it returns 1 if the conversion is not possible.

> **Note:** You can use this function in a WHERE clause of a SQL statement, as shown in the example.

Syntax

```
DBMS_ROWID.ROWID_VERIFY (
   rowid_in        IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_verify,WNDS,WNPS,RNPS);
```

Parameters

Table 62–12 ROWID_VERIFY Function Parameters

Parameter	Description
rowid_in	ROWID to be verified.
schema_name	Name of the schema which contains the table.
object_name	Table name.
conversion_type	rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type or the character string).

Example

Considering the schema in the example for the ROWID_TO_EXTENDED function, you can use the following statement to find bad ROWIDs prior to conversion. This enables you to fix them beforehand.

```
SELECT ROWID, rowid_col
   FROM SCOTT.RIDS
   WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;
```

See Also: Chapter 98, "UTL_RAW", Chapter 99, "UTL_REF"

63

DBMS_RULE

The DBMS_RULE package contains the EVALUATE procedure.

This chapter contains the following topic:

- Summary of DBMS_RULE Subprograms

> **Note:** PUBLIC is granted execute privilege on this package.

See Also:

- Chapter 109, "Rule Types" for more information about the types used with the DBMS_RULE package
- Chapter 64, "DBMS_RULE_ADM" and *Oracle9i Streams* for more information about rules

Summary of DBMS_RULE Subprograms

Table 63–1 DBMS_RULE Subprogram

Subprogram	Description
"EVALUATE Procedure" on page 63-3	Evaluates the rules in the specified rule set that use the evaluation context specified

EVALUATE Procedure

Evaluates the rules in the specified rule set that use the evaluation context specified.

> **Note:** Rules in the rule set that use an evaluation context different from the one specified are not considered for evaluation.

The rules in the rule set are evaluated using the data specified for `table_values`, `column_values`, `variable_values`, and `attribute_values`. These values must refer to tables and variables in the specified evaluation context. Otherwise, they are ignored.

The caller may specify, using `stop_on_first_hit`, if evaluation must stop as soon as the first TRUE rule or the first MAYBE rule (if there are no TRUE rules) is found.

The caller may also specify, using `simple_rules_only`, if only rules that are simple enough to be evaluated fast (which means without SQL) should be considered for evaluation. This makes evaluation faster, but causes rules that cannot be evaluated without SQL to be returned as MAYBE rules.

Partial evaluation is supported. The EVALUATE procedure can be called with data for only some of the tables, columns, variables, or attributes. In such a case, rules that cannot be evaluated because of a lack of data are returned as MAYBE rules, unless they can be determined to be TRUE or FALSE based on the values of one or more simple expressions within the rule. For example, given a value of `1` for attribute `"a.b"` of variable `"x"`, a rule with the following rule condition can be returned as TRUE, without a value for table `"tab"`:

```
(:x.a.b = 1) or (tab.c > 10)
```

The results of an evaluation are the following:

- TRUE rules, which is the list of rules that evaluate to TRUE based on the given data. These rules are returned in the OUT parameter true_rules.
- MAYBE rules, which is the list of rules that could not be evaluated for one of the following reasons:
  - The rule refers to data that was unavailable. For example, a variable attribute "x.a.b" is specified, but no value is specified for the variable "x", the attribute "a", or the attribute "a.b".
  - The rule is not simple enough to be evaluated fast (without SQL) and simple_rules_only is specified as TRUE.

  MAYBE rules are returned in the OUT parameter maybe_rules.

To run this procedure, a user must meet at least one of the following requirements:

- Have EXECUTE_ON_RULE_SET privilege on the rule set
- Have EXECUTE_ANY_RULE_SET system privilege
- Be the rule set owner

Note: The rules engine does not invoke any actions. An action context can be returned with each returned rule, but the client of the rules engine must invoke any necessary actions.

See Also: Chapter 109, "Rule Types" for more information about the types used with the DBMS_RULE package

Syntax

```
DBMS_RULE.EVALUATE(
  rule_set_name         IN   VARCHAR2,
  evaluation_context    IN   VARCHAR2,
  event_context         IN   SYS.RE$NV_LIST               DEFAULT NULL,
  table_values          IN   SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
  column_values         IN   SYS.RE$COLUMN_VALUE_LIST,
  variable_values       IN   SYS.RE$VARIABLE_VALUE_LIST   DEFAULT NULL,
  attribute_values      IN   SYS.RE$ATTRIBUTE_VALUE_LIST,
  stop_on_first_hit     IN   BOOLEAN                      DEFAULT FALSE,
  simple_rules_only     IN   BOOLEAN                      DEFAULT FALSE,
  true_rules            OUT  SYS.RE$RULE_HIT_LIST,
  maybe_rules           OUT  SYS.RE$RULE_HIT_LIST);
```

Note: This procedure is overloaded. One version of this procedure has the `column_values` and `attribute_values` parameters, and the other does not.

Parameters

Table 63–2 EVALUATE Procedure Parameters **(Page 1 of 2)**

Parameter	Description
rule_set_name	Name of the rule set in the form [*schema_name.*]*rule_set_name*. For example, to evaluate all of the rules in a rule set named hr_rules in the hr schema, enter hr.hr_rules for this parameter. If the schema is not specified, then the schema of the current user is used.
evaluation_context	An evaluation context name in the form [*schema_name.*]*evaluation_context_name*. If the schema is not specified, then the name of the current user is used. Only rules that use the specified evaluation context are evaluated.
event_context	A list of name-value pairs that identify events that cause evaluation
table_values	Contains the data for table rows using the table aliases specified when the evaluation context was created
column_values	Contains the partial data for table rows. It must not contain column values for tables, whose values are already specified in table_values.
variable_values	A list containing the data for variables. The only way for an explicit variable value to be known is to specify its value in this list. If an implicit variable value is not specified in the list, then the function used to obtain the value of the implicit variable is invoked. If an implicit variable value is specified in the list, then this value is used and the function is not invoked.
attribute_values	Contains the partial data for variables. It must not contain attribute values for variables whose values are already specified in variable_values.
stop_on_first_hit	If TRUE, then the rules engine stops evaluation as soon as it finds a TRUE rule. If TRUE and there are no TRUE rules, then the rules engine stops evaluation as soon as it finds a rule that may evaluate to TRUE given more data. If FALSE, then the rules engine continues to evaluate rules even after it finds a TRUE rule.

Table 63–2 EVALUATE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
simple_rules_only	If TRUE, then only those rules that are simple enough to be evaluated fast (without issuing SQL) are considered for evaluation. If FALSE, then evaluates all rules.
true_rules	Receives the output of the EVALUATE procedure into a varray of RE$RULE_HIT_LIST type. If no rules evaluate to TRUE, then true_rules is empty. If at least one rule evaluates to TRUE and stop_on_first_hit is TRUE, then true_rules contains one rule that evaluates to TRUE. If stop_on_first_hit is FALSE, then true_rules contains all rules that evaluate to TRUE.
maybe_rules	If all rules can be evaluated completely, without requiring any additional data, then maybe_rules is empty. If stop_on_first_hit is TRUE, then if there is at least one rule that may evaluate to TRUE given more data, and no rules evaluate to TRUE, then maybe_rules contains one rule that may evaluate to TRUE. If stop_on_first_hit is FALSE, then maybe_rules contains all rules that may evaluate to TRUE given more data.

64

DBMS_RULE_ADM

The DBMS_RULE_ADM package provides the administrative interface for creating and managing rules, rule sets, and rule evaluation contexts.

This chapter contains the following topic:

- Summary of DBMS_RULE_ADM Subprograms

> **Note:** PUBLIC is granted execute privilege on this package.

See Also:

- Chapter 109, "Rule Types" for more information about the types used with the DBMS_RULE_ADM package
- Chapter 63, "DBMS_RULE" and *Oracle9i Streams* for more information about rules

Summary of DBMS_RULE_ADM Subprograms

Table 64–1 DBMS_RULE_ADM Subprograms

Subprogram	Description
"ADD_RULE Procedure" on page 64-3	Adds the specified rule to the specified rule set
"ALTER_RULE Procedure" on page 64-5	Changes one or more aspects of the specified rule
"CREATE_EVALUATION_CONTEXT Procedure" on page 64-8	Creates a rule evaluation context
"CREATE_RULE Procedure" on page 64-11	Creates a rule with the specified name
"CREATE_RULE_SET Procedure" on page 64-13	Creates a rule set with the specified name
"DROP_EVALUATION_CONTEXT Procedure" on page 64-14	Drops the rule evaluation context with the specified name
"DROP_RULE Procedure" on page 64-15	Drops the rule with the specified name
"DROP_RULE_SET Procedure" on page 64-16	Drops the rule set with the specified name
"GRANT_OBJECT_PRIVILEGE Procedure" on page 64-17	Grants the specified object privilege on the specified object to the specified user or role
"GRANT_SYSTEM_PRIVILEGE Procedure" on page 64-20	Grants the specified system privilege to the specified user or role
"REMOVE_RULE Procedure" on page 64-23	Removes the specified rule from the specified rule set
"REVOKE_OBJECT_PRIVILEGE Procedure" on page 64-25	Revokes the specified object privilege on the specified object from the specified user or role
"REVOKE_SYSTEM_PRIVILEGE Procedure" on page 64-26	Revokes the specified system privilege from the specified user or role

Note: All procedures commit unless specified otherwise.

ADD_RULE Procedure

Adds the specified rule to the specified rule set.

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE_SET privilege on the rule set
- Have ALTER_ANY_RULE_SET system privilege
- Be the owner of the rule set

Also, the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_RULE privilege on the rule
- Have EXECUTE_ANY_RULE system privilege
- Be the rule owner

If the rule has no evaluation context and no evaluation context is specified when you run this procedure, then rule uses the evaluation context associated with the rule set. In such a case, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

If the evaluation context owner is different than the rule owner, then the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

Syntax

```
DBMS_RULE_ADM.ADD_RULE(
   rule_name           IN  VARCHAR2,
   rule_set_name       IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2   DEFAULT NULL,
   rule_comment        IN  VARCHAR2   DEFAULT NULL);
```

Parameters

Table 64–2 ADD_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are adding to the rule set, specified as [*schema_name.*]*rule_name*. For example, to add a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set to which you are adding the rule, specified as [*schema_name.*]*rule_set_name*. For example, to add the rule to a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context	An evaluation context name in the form [*schema_name.*]*evaluation_context_name*. If the schema is not specified, then the current user is the default. Only specify an evaluation context if the rule itself does not have an evaluation context and you do not want to use the rule set's evaluation context for the rule.
rule_comment	Optional description, which may contain the reason for adding the rule to the rule set

ALTER_RULE Procedure

Changes one or more aspects of the specified rule.

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE privilege on the rule
- Have ALTER_ANY_RULE system privilege
- Be the owner of the rule being altered

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

If the evaluation context owner is different than the rule owner, then the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

> **See Also:** Chapter 109, "Rule Types" for more information about the types used with the DBMS_RULE_ADM package

Syntax

```
DBMS_RULE_ADM.ALTER_RULE(
   rule_name                 IN  VARCHAR2,
   condition                 IN  VARCHAR2         DEFAULT NULL,
   evaluation_context        IN  VARCHAR2         DEFAULT NULL,
   remove_evaluation_context IN  BOOLEAN          DEFAULT FALSE,
   action_context            IN  SYS.RE$NV_LIST   DEFAULT NULL,
   remove_action_context     IN  BOOLEAN          DEFAULT FALSE,
   rule_comment              IN  VARCHAR2         DEFAULT NULL,
   remove_rule_comment       IN  BOOLEAN          DEFAULT FALSE);
```

Parameters

Table 64–3 ALTER_RULE Procedure Parameters **(Page 1 of 2)**

Parameter	Description
rule_name	The name of the rule you are altering, specified as [*schema_name.*]*rule_name*. For example, to alter a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
condition	The Boolean condition to be associated with the rule. If non-NULL, then the rule's condition is changed.
evaluation_context	An evaluation context name in the form [*schema_name.*]*evaluation_context_name*. If the schema is not specified, then the current user is the default. If non-NULL, then the rule's evaluation context is changed.
remove_evaluation_context	If true, then sets the evaluation context for the rule to NULL, which effectively removes the evaluation context from the rule. If false, then retains any evaluation context for the specified rule. If the evaluation_context parameter is non-NULL, then this parameter should be set to false.
action_context	If non-NULL, then changes the action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
remove_action_context	If true, then sets the action context for the rule to NULL, which effectively removes the action context from the rule. If false, then retains any action context for the specified rule. If the action_context parameter is non-NULL, then this parameter should be set to false.

Table 64–3 ALTER_RULE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
rule_comment	If non-NULL, then changes the description of the rule
remove_rule_comment	If true, then sets the comment for the rule to NULL, which effectively removes the comment from the rule. If false, then retains any comment for the specified rule. If the rule_comment parameter is non-NULL, then this parameter should be set to false.

CREATE_EVALUATION_CONTEXT Procedure

Creates a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being created and have `CREATE_EVALUATION_CONTEXT_OBJ` system privilege
- Have `CREATE_ANY_EVALUATION_CONTEXT` system privilege

> **See Also:** Chapter 109, "Rule Types" for more information about the types used with the `DBMS_RULE_ADM` package

Syntax

```
DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
  evaluation_context_name      IN  VARCHAR2,
  table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST    DEFAULT NULL,
  variable_types               IN  SYS.RE$VARIABLE_TYPE_LIST  DEFAULT NULL,
  evaluation_function          IN  VARCHAR2                   DEFAULT NULL,
  evaluation_context_comment   IN  VARCHAR2                   DEFAULT NULL);
```

Parameters

Table 64–4 CREATE_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
evaluation_context_name	The name of the evaluation context you are creating, specified as [*schema_name.*]*evaluation_context_name*. For example, to create an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.
table_aliases	Table aliases that specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.
variable_types	A list of variables containing the explicit and implicit variables for the evaluation context
evaluation_function	An optional function that will be called to evaluate rules using the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default. See "Usage Notes" for more information about the evaluation function.
evaluation_context_comment	An optional description of the rule evaluation context.

Usage Notes

The evaluation function must have the following signature:

```
FUNCTION evaluation_function_name(
  rule_set_name       IN   VARCHAR2,
  evaluation_context  IN   VARCHAR2,
  event_context       IN   SYS.RE$NV_LIST               DEFAULT NULL,
  table_values        IN   SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
  column_values       IN   SYS.RE$COLUMN_VALUE_LIST     DEFAULT NULL,
  variable_values     IN   SYS.RE$VARIABLE_VALUE_LIST   DEFAULT NULL,
  attribute_values    IN   SYS.RE$ATTRIBUTE_VALUE_LIST  DEFAULT NULL,
  stop_on_first_hit   IN   BOOLEAN                      DEFAULT FALSE,
  simple_rules_only   IN   BOOLEAN                      DEFAULT FALSE,
  true_rules          OUT  SYS.RE$RULE_HIT_LIST,
  maybe_rules         OUT  SYS.RE$RULE_HIT_LIST);
RETURN BINARY_INTEGER;
```

Note: Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.

The return value of the function must be one of the following:

- DBMS_RULE_ADM.EVALUATION_SUCCESS
- DBMS_RULE_ADM.EVALUATION_FAILURE
- DBMS_RULE_ADM.EVALUATION_CONTINUE

CREATE_RULE Procedure

Creates a rule.

To run this procedure, a user must meet at least one of the following requirements:

- The user must be the owner of the rule being created and the CREATE_RULE_OBJ system privilege.
- The user must have CREATE_ANY_RULE system privilege.

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS.
- Be the evaluation context owner

If the evaluation context owner is different than the rule owner, then the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

> **See Also:** Chapter 109, "Rule Types" for more information about the types used with the DBMS_RULE_ADM package

Syntax

```
DBMS_RULE_ADM.CREATE_RULE(
   rule_name           IN  VARCHAR2,
   condition           IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2        DEFAULT NULL,
   action_context      IN  SYS.RE$NV_LIST  DEFAULT NULL,
   rule_comment        IN  VARCHAR2        DEFAULT NULL);
```

Parameters

Table 64–5 CREATE_RULE Procedure Parameters

Parameter	Description
`rule_name`	The name of the rule you are creating, specified as `[`*`schema_name`*`.]`*`rule_name`*. For example, to create a rule named `all_a` in the `hr` schema, enter `hr.all_a` for this parameter. If the schema is not specified, then the current user is the default.
`condition`	The Boolean condition to be associated with the rule. A Boolean condition evaluates to `TRUE` or `FALSE` and can be any condition allowed in the `WHERE` clause of a `SELECT` statement. For example, the following is a valid rule condition: `department_id = 30` **Note:** Do not include the word `"WHERE"` in the condition.
`evaluation_context`	An optional evaluation context name in the form `[`*`schema_name`*`.]`*`evaluation_context_name`*, which is associated with the rule. If the schema is not specified, then the current user is the default. If `evaluation_context` is not specified, then the rule inherits the evaluation context from its rule set.
`action_context`	The action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
`rule_comment`	An optional description of the rule

CREATE_RULE_SET Procedure

Creates a rule set.

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule set being created and have CREATE_RULE_SET_OBJ system privilege
- Have CREATE_ANY_RULE_SET system privilege

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

Syntax

```
DBMS_RULE_ADM.CREATE_RULE_SET(
   rule_set_name       IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2  DEFAULT NULL,
   rule_set_comment    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 64–6 CREATE_RULE_SET Procedure Parameters

Parameter	Description
rule_set_name	The name of the rule set you are creating, specified as [*schema_name.*]*rule_set_name*. For example, to create a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context	An optional evaluation context name in the form [*schema_name.*]*evaluation_context_name*, which applies to all rules in the rule set that are not associated with an evaluation context explicitly. If the schema is not specified, then the current user is the default.
rule_set_comment	An optional description of the rule set

DROP_EVALUATION_CONTEXT Procedure

Drops a rule evaluation context.

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context
- Have DROP_ANY_EVALUATION_CONTEXT system privilege

Syntax

```
DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT(
   evaluation_context_name  IN  VARCHAR2,
   force                    IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 64–7 DROP_EVALUATION_CONTEXT Procedure Parameters

Parameter	Description
evaluation_context_name	The name of the evaluation context you are dropping, specified as [*schema_name*.]*evaluation_context_name*. For example, to drop an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.
force	If true, then removes the rule evaluation context from all rules and rule sets that use it. If false and no rules or rule sets use the rule evaluation context, then drops the rule evaluation context. If false and one or more rules or rule sets use the rule evaluation context, then raises an exception. **Caution:** Setting force to true can result in rules and rule sets that do not have an evaluation context. If neither a rule nor the rule set it is in has an evaluation context, and no evaluation context was specified for the rule by the ADD_RULE procedure, then the rule cannot be evaluated.

DROP_RULE Procedure

Drops a rule.

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule
- Have DROP_ANY_RULE system privilege

Note:

- To remove a rule from a rule set without dropping the rule from the database, use the REMOVE_RULE procedure.
- The rule evaluation context associated with the rule, if any, is not dropped when you run this procedure.

Syntax

```
DBMS_RULE_ADM.DROP_RULE(
   rule_name  IN  VARCHAR2,
   force      IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 64–8 DROP_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are dropping, specified as [*schema_name.*]*rule_name*. For example, to drop a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
force	If TRUE, then removes the rule from all rule sets that contain it. If FALSE and no rule sets contain the rule, then drops the rule. If FALSE and one or more rule sets contain the rule, then raises an exception.

DROP_RULE_SET Procedure

Drops a rule set.

To run this procedure, a user must meet at least one of the following requirements:

- Have DROP_ANY_RULE_SET system privilege
- Be the owner of the rule set

> **Note:** The rule evaluation context associated with the rule set, if any, is not dropped when you run this procedure.

Syntax

```
DBMS_RULE_ADM.DROP_RULE_SET(
   rule_set_name    IN  VARCHAR2,
   delete_rules     IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 64–9 DROP_RULE_SET Procedure Parameters

Parameter	Description
rule_set_name	The name of the rule set you are dropping, specified as [*schema_name.*]*rule_set_name*. For example, to drop a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
delete_rules	If TRUE, then also drops any rules that are in the rule set. If any of the rules in the rule set are also in another rule set, then these rules are not dropped. If FALSE, then the rules in the rule set are retained.

GRANT_OBJECT_PRIVILEGE Procedure

Grants the specified object privilege on the specified object to the specified user or role. If a user owns the object, then the user automatically is granted all privileges on the object, with grant option.

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

In addition, if the object is a rule set, then the user must have EXECUTE privilege on all the rules in the rule set with grant option or must own the rules in the rule set.

Syntax

```
DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
   privilege     IN  BINARY_INTEGER,
   object_name   IN  VARCHAR2,
   grantee       IN  VARCHAR2,
   grant_option  IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 64–10 GRANT_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the object privilege to grant to the grantee on the object. See "Usage Notes" on page 64-19 for the available object privileges.
`object_name`	The name of the object for which you are granting the privilege to the grantee, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, to grant the privilege on a rule set named `apply_rules` in the `hr` schema, enter `hr.apply_rules` for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
`grantee`	The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.
`grant_option`	If `true`, then the specified user or users granted the specified privilege can grant this privilege to others. If `false`, then the specified user or users granted the specified privilege cannot grant this privilege to others.

Usage Notes

Table 64–11 lists the object privileges.

Table 64–11 Object Privileges for Evaluation Contexts, Rules, and Rule Sets

Privilege	Description
SYS.DBMS_RULE_ADM.ALL_ON_EVALUATION_CONTEXT	Alter and execute a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.ALL_ON_RULE	Alter and execute a particular rule in another user's schema
SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET	Alter and execute a particular rule set in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_EVALUATION_CONTEXT	Alter a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_RULE	Alter a particular rule in another user's schema
SYS.DBMS_RULE_ADM.ALTER_ON_RULE_SET	Alter a particular rule set in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_EVALUATION_CONTEXT	Execute a particular evaluation context in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE	Execute a particular rule in another user's schema
SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET	Execute a particular rule set in another user's schema

For example, to grant the `hr` user the privilege to alter a rule named `hr_dml` in the `strmadmin` schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALTER_ON_RULE,
    object_name  => 'strmadmin.hr_dml',
    grantee      => 'hr',
    grant_option => false);
END;
/
```

GRANT_SYSTEM_PRIVILEGE Procedure

Grants the specified system privilege to the specified user or role.

Syntax

```
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
   privilege     IN  BINARY_INTEGER,
   grantee       IN  VARCHAR2,
   grant_option  IN  BOOLEAN   DEFAULT false);
```

Parameters

Table 64–12 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the system privilege to grant to the grantee. See "Usage Notes" on page 64-21 for the available system privileges.
`grantee`	The name of the user or role for which the privilege is granted
`grant_option`	If `true`, then the specified user or users granted the specified privilege can grant the system privilege to others. If `false`, then the specified user or users granted the specified privilege cannot grant the system privilege to others.

Usage Notes

Table 64–13 lists the system privileges.

Table 64–13 System Privileges for Evaluation Contexts, Rules, and Rule Sets

Privilege	Description
SYS.DBMS_RULE_ADM.ALTER_ANY_EVALUATION_CONTEXT	Alter any evaluation context owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE	Alter any rule owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE_SET	Alter any rule set owned by any user
SYS.DBMS_RULE_ADM.CREATE_ANY_EVALUATION_CONTEXT	Create a new evaluation context in any schema
SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ	Create a new evaluation context in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE	Create a new rule in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_OBJ	Create a new rule in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET	Create a new rule set in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_SET_OBJ	Create a new rule set in the grantee's schema
SYS.DBMS_RULE_ADM.DROP_ANY_EVALUATION_CONTEXT	Drop any evaluation context in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE	Drop any rule in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE_SET	Drop any rule set in any schema
SYS.DBMS_RULE_ADM.EXECUTE_ANY_EVALUATION_CONTEXT	Execute any evaluation context owned by any user
SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE	Execute any rule owned by any user
SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE_SET	Execute any rule set owned by any user

For example, to grant the strmadmin user the privilege to create a rule set in any schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET,
    grantee      => 'strmadmin',
    grant_option => false);
END;
/
```

Note: When you grant a privilege on "ANY" object (for example, ALTER_ANY_RULE), and the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE, you give the user access to that type of object in all schemas, except the SYS schema. By default, the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE.

If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to TRUE. Then privileges granted on "ANY" object will allow access to any schema, including SYS.

REMOVE_RULE Procedure

Removes the specified rule from the specified rule set.

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE_SET privilege on the rule set
- Have ALTER_ANY_RULE_SET system privilege
- Be the owner of the rule set

> **Note:** This procedure does not drop a rule from the database. To drop a rule from the database, use the DROP_RULE procedure.

Syntax

```
DBMS_RULE_ADM.REMOVE_RULE(
   rule_name               IN VARCHAR2,
   rule_set_name           IN VARCHAR2,
   evaluation_context      IN VARCHAR2  DEFAULT NULL,
   all_evaluation_contexts IN BOOLEAN   DEFAULT false);
```

Parameters

Table 64–14 REMOVE_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule you are removing from the rule set, specified as [*schema_name.*]*rule_name*. For example, to remove a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set from which you are removing the rule, specified as [*schema_name.*]*rule_set_name*. For example, to remove the rule from a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context_name	The name of the evaluation context associated with the rule you are removing, specified as [*schema_name.*]*evaluation_context_name*. For example, to specify an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default. If an evaluation context was specified for the rule you are removing when you added the rule to the rule set using the ADD_RULE procedure, then specify the same evaluation context. If you added the same rule more than once with different evaluation contexts, then specify the rule with the evaluation context you want to remove. If you specify an evaluation context that is not associated with the rule, then an error is raised. Specify NULL if you did not specify an evaluation context when you added the rule to the rule set. If you specify NULL and there are one or more evaluation contexts associated with the rule, then an error is raised.
all_evaluation_contexts	If true, then the rule is removed from the rule set with all of its associated evaluation contexts. If false, then only the rule with the specified evaluation context is removed. This parameter is relevant only if the same rule is added more than once to the rule set with different evaluation contexts.

REVOKE_OBJECT_PRIVILEGE Procedure

Revokes the specified object privilege on the specified object from the specified user or role.

Syntax

```
DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE(
   privilege     IN  BINARY_INTEGER,
   object_name   IN  VARCHAR2,
   revokee       IN  VARCHAR2);
```

Parameters

Table 64–15 REVOKE_OBJECT_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the object privilege on the object to revoke from the revokee. See "GRANT_OBJECT_PRIVILEGE Procedure" on page 64-17 for a list of the object privileges.
`object_name`	The name of the object for which you are revoking the privilege from the revokee, specified as `[`*`schema_name`*`.]`*`object_name`*. For example, to revoke an object privilege on a rule set named `apply_rules` in the `hr` schema, enter `hr.apply_rules` for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
`revokee`	The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.

REVOKE_SYSTEM_PRIVILEGE Procedure

Revokes the specified system privilege from the specified user or role.

Syntax

```
DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(
   privilege  IN  BINARY_INTEGER,
   revokee    IN  VARCHAR2);
```

Parameters

Table 64–16 REVOKEE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
`privilege`	The name of the system privilege to revoke from the revokee. See "GRANT_SYSTEM_PRIVILEGE Procedure" on page 64-20 for a list of the system privileges.
`revokee`	The name of the user or role from which the privilege is revoked

65

DBMS_SESSION

This package provides access to SQL `ALTER SESSION` and `SET ROLE` statements, and other session information, from PL/SQL. You can use this to set preferences and security levels.

This chapter discusses the following topics:

- Requirements
- Summary of DBMS_SESSION Subprograms

Requirements

This package runs with the privileges of the calling user, rather than the package owner SYS.

Summary of DBMS_SESSION Subprograms

Table 65–1 DBMS_SESSION Subprograms

Subprogram	Description
SET_IDENTIFIER on page 65-3	Sets the identifier.
SET_CONTEXT on page 65-4 and on page 65-4	Sets the context.
CLEAR_CONTEXT on page 65-5	Clears the context.
CLEAR_IDENTIFIER on page 65-6	Clears the identifier.
SET_ROLE Procedure on page 65-7	Sets role.
SET_SQL_TRACE Procedure on page 65-7	Turns tracing on or off.
SET_NLS Procedure on page 65-8	Sets national language support (NLS).
CLOSE_DATABASE_LINK Procedure on page 65-8	Closes database link.
RESET_PACKAGE Procedure on page 65-9	Deinstantiates all packages in the session.
MODIFY_PACKAGE_STATE Procedure on page 65-10	Performs actions on the session state of PL/SQL program units that are active in the session.
UNIQUE_SESSION_ID Function on page 65-14	Returns an identifier that is unique for all sessions currently connected to this database.
IS_ROLE_ENABLED Function on page 65-14	Determines if the named role is enabled for the session.
IS_SESSION_ALIVE Function on page 65-15	Determines if the specified session is active.
SET_CLOSE_CACHED_OPEN_CURSORS Procedure on page 65-16	Turns `close_cached_open_cursors` on or off.
FREE_UNUSED_USER_MEMORY Procedure on page 65-16	Lets you reclaim unused memory after performing operations requiring large amounts of memory.

Table 65–1 DBMS_SESSION Subprograms

Subprogram	Description
SET_CONTEXT Procedure on page 65-19	Sets or resets the value of a context attribute.
LIST_CONTEXT Procedure on page 65-19	Returns a list of active namespace and context for the current session.
SWITCH_CURRENT_CONSUMER_GROUP Procedure on page 65-20	Facilitates changing the current resource consumer group of a user's current session.

SET_IDENTIFIER

This procedure sets the client ID in the session.

Syntax

```
DBMS_SESSION.SET_IDENTIFIER (
   client_id VARCHAR2);
```

Parameters

Table 65–2 SET_IDENTIFIER Procedure Parameters

Parameter	Description
client_id	The application-specific identifier of the current database session.

Usage Notes

Note the following:

- SET_IDENTIFIER initializes the current session with a client identifier to identify the associated global application context
- client_id is case sensitive; it must match the client_id parameter in the set_context
- This procedure is executable by public

SET_CONTEXT

This procedure sets the context.

Syntax

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
   value     VARCHAR2);
```

Parameters

Table 65–3 SET_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace of the application context to be set
attribute	The attribute of the application context to be set
value	The value of the application context to be set

Usage Notes

Note the following:

- This interface is maintained for 8i compatibility
- If the namespace is a global context namespace, then `username` is assigned the current user name, and `client_id` will be assigned the current `client_id` in the session; NULL if not set.
- This procedure must be invoked directly or indirectly by the trusted package

SET_CONTEXT Procedure

This procedure sets the context.

Syntax

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
```

```
   value     VARCHAR2,
   username  VARCHAR2,
   client_id VARCHAR2 );
```

Parameters

Table 65–4 SET_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace of the application context to be set
attribute	The attribute of the application context to be set
value	The value of the application context to be set
username	The username attribute of the application context
client_id	The client_id attribute of the application context (64-byte maximum)

Usage Notes

Note the following:

- Sets the application context and associates it with the client_id
- Username must be a valid SQL identifier
- client_id is a string of at most 64 bytes
- client_id is case sensitive; it must match the argument to set_identifier
- Must be invoked directly or indirectly by the trusted package
- Can only be used on global namespaces

CLEAR_CONTEXT

Syntax

```
DBMS_SESSION.CLEAR_CONTEXT
   namespace         VARCHAR2,
   client_identifier VARCHAR2
   attribute         VARCHAR2);
```

Parameters

Table 65–5 CLEAR_CONTEXT Procedure Parameters

Parameter	Description
namespace	The namespace in which the application context is to be cleared. Required. For a session-local context, namespace must be specified. If namespace is defined as Session Local Context, then client_identifier is optional since it is only associated with a globally accessed context. For a globally accessed context, namespace must be specified. NULL is a valid value for client_identifier because a session with no identifier set can see a context that looks like the (namespace,attribute,value,username,null) set using SET_CONTEXT.
client_ identifier	Applies to a global context and is optional for other types of contexts; 64-byte maximum.
attribute	The specific attribute in the namespace to be cleared. Optional. the default is NULL. If you specify attribute as NULL, then (namespace,attribute,value) for that namespace are cleared from the session. If attribute is not specified, then all context information that has the namespace and client_identifier arguments is cleared.

Usage Notes

This procedure must be invoked directly or indirectly by the trusted package.

CLEAR_IDENTIFIER

This procedure removes the set_client_id in the session.

Syntax

```
DBMS_SESSION.CLEAR_IDENTIFIER();
```

Usage Notes

This procedure is executable by public.

SET_ROLE Procedure

This procedure enables and disables roles. It is equivalent to the SET ROLE SQL statement.

Syntax

```
DBMS_SESSION.SET_ROLE (
   role_cmd VARCHAR2);
```

Parameters

Table 65–6 SET_ROLE Procedure Parameters

Parameter	Description
role_cmd	This text is appended to "set role" and then run as SQL.

SET_SQL_TRACE Procedure

This procedure turns tracing on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET SQL_TRACE ...
```

Syntax

```
DBMS_SESSION.SET_SQL_TRACE (
   sql_trace boolean);
```

Parameters

Table 65–7 SET_SQL_TRACE Procedure Parameters

Parameter	Description
sql_trace	TRUE turns tracing on, FALSE turns tracing off.

SET_NLS Procedure

This procedure sets up your national language support (NLS). It is equivalent to the following SQL statement:

```
ALTER SESSION SET <nls_parameter> = <value>
```

Syntax

```
DBMS_SESSION.SET_NLS (
   param VARCHAR2,
   value VARCHAR2);
```

Parameters

Table 65–8 SET_NLS Procedure Parameters

Parameter	Description
`param`	NLS parameter. The parameter name must begin with 'NLS'.
`value`	Parameter value. If the parameter is a text literal, then it needs embedded single-quotes. For example, "`set_nls('nls_date_format','''DD-MON-YY''')`"

CLOSE_DATABASE_LINK Procedure

This procedure closes an open database link. It is equivalent to the following SQL statement:

```
ALTER SESSION CLOSE DATABASE LINK <name>
```

Syntax

```
DBMS_SESSION.CLOSE_DATABASE_LINK (
   dblink VARCHAR2);
```

Parameters

Table 65–9 CLOSE_DATABASE_LINK Procedure Parameters

Parameter	Description
dblink	Name of the database link to close.

RESET_PACKAGE Procedure

This procedure deinstantiates all packages in this session: It frees all package states. See "MODIFY_PACKAGE_STATE Procedure" on page 65-10.

Memory used for caching execution state is associated with all PL/SQL functions, procedures, and packages that have been run in a session.

For packages, this collection of memory holds the current values of package variables and controls the cache of cursors opened by the respective PL/SQL programs. A call to RESET_PACKAGE frees the memory associated with each of the previously run PL/SQL programs from the session, and, consequently, clears the current values of any package globals and closes any cached cursors.

RESET_PACKAGE can also be used to reliably restart a failed program in a session. If a program containing package variables fails, then it is hard to determine which variables need to be reinitialized. RESET_PACKAGE guarantees that all package variables are reset to their initial values.

Syntax

```
DBMS_SESSION.RESET_PACKAGE;
```

Usage Notes

Because the amount of memory consumed by all executed PL/SQL can become large, you might use RESET_PACKAGE to trim down the session memory footprint at certain points in your database application. However, make sure that resetting package variable values will not affect the application. Also, remember that later execution of programs that have lost their cached memory and cursors will perform slower, because they need to re-create the freed memory and cursors.

RESET_PACKAGE does not free the memory, cursors, and package variables immediately when called.

Note: RESET_PACKAGE only frees the memory, cursors, and package variables *after* the PL/SQL call that made the invocation finishes running.

For example, PL/SQL procedure P1 calls PL/SQL procedure P2, and P2 calls RESET_PACKAGE. The RESET_PACKAGE effects do not occur until procedure P1 finishes execution (the PL/SQL call ends).

Example

This SQL*Plus script runs a large program with many PL/SQL program units that may or may not use global variables, but it doesn't need them beyond this execution:

```
EXCECUTE large_plsql_program1;
```

To free up PL/SQL cached session memory:

```
EXECUTE DBMS_SESSION.RESET_PACKAGE;
```

To run another large program:

```
EXECUTE large_plsql_program2;
```

MODIFY_PACKAGE_STATE Procedure

This procedure performs actions on the session state of PL/SQL program units that are active in the session. The procedure uses the DBMS_SESSION constants shown in Table 65–10.

Because the client-side PL/SQL code cannot reference remote package variables or constants, you must explicitly use the values of the constants. For example, the following code does not compile on the client because it uses the constant DBMS_SESSION.REINITIALIZE:

```
DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE);
```

Instead, use the following code on the client, because the argument is explicitly provided:

```
DBMS_SESSION.MODIFY_PACKAGE_STATE(2) -- compiles on the client
```

DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.FREE_ALL_RESOURCES)behaves identically to DBMS_SESSION.RESET_PACKAGE. You should use DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.FREE_ALL_RESOURCES) instead of DBMS_SESSION.RESET_PACKAGE.

Syntax

```
DBMS_SESSION.MODIFY_PACKAGE_STATE(
   action_flags IN PLS_INTEGER);
```

Constants

See "Usage Notes" on page 65-12 for differences between the flags and why DBMS_SESSION.REINITIALIZE should exhibit better performance than DBMS_SESSION.FREE_ALL_RESOURCES.

Table 65–10 action_flags Constants for MODIFY_PACKAGE_STATE

Constant	Description
FREE_ALL_RESOURCES	PLS_INTEGER := 1
REINITIALIZE	PLS_INTEGER := 2

Parameters

Table 65–11 MODIFY_PACKAGE_STATE Procedure Parameters

Parameter	Description
action_flags	Bit flags that determine the action taken on PL/SQL program units: ▪ FREE_ALL_RESOURCES (or 1)—frees all memory associated with each of the previously run PL/SQL programs from the session. Clears the current values of any package globals and closes cached cursors. On subsequent use, the PL/SQL program units are reinstantiated and package globals are reinitialized. ▪ REINITIALIZE (or 2)—reinitializes packages without actually being freed and re-created from scratch. Instead the package memory is reused.

Usage Notes

- For both FREE_ALL_RESOURCES and REINITIALIZE, reinitialization takes effect after the PL/SQL call that made the current invocation finishes running.
- Reinitialization occurs only if the package is actually referenced. Packages are reinitialized in the order in which they are referenced.
- REINITIALIZE differs from FREE_ALL_RESOURCES in that any open cursors are closed, semantically speaking. However, the cursor resource is not actually freed. It is returned to the PL/SQL cursor cache. The cursor cache is not flushed. Hence, cursors corresponding to frequently accessed static SQL in PL/SQL will remain cached in the PL/SQL cursor cache and the application will not incur the overhead of opening, parsing, and closing a new cursor for those statements on subsequent use.
- The session memory for PL/SQL modules without a global state (such as types or stored procedures) is not freed and re-created.
- When using FREE_ALL_RESOURCES or REINITIALIZE, make sure that resetting package variable values does not affect the application.
- Because DBMS_SESSION.REINITIALIZE does not actually cause all the package state to be freed, in some situations, the application could use significantly more session memory than if the FREE_ALL_RESOURCES flag or the RESET_PACKAGE procedure had been used. For instance, after performing DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE), if the application does not refer to many of the packages that were previously referenced, then the session memory for those packages will remain until the end of the session (or until DBMS_SESSION.RESET_PACKAGE is called).

Using DBMS_SESSION.MODIFY_PACKAGE_STATE: Example

This example illustrates the use of DBMS_SESSION.MODIFY_PACKAGE_STATE. Consider a package P with some global state (a cursor c and a number cnt). When the package is first initialized, the package variable cnt is 0 and the cursor c is CLOSED. Then, in the session, change the value of cnt to 111 and also execute an OPEN operation on the cursor. If you call print_status to display the state of the package, you see that cnt is 111 and that the cursor is OPEN. Next, call DBMS_SESSION.MODIFY_PACKAGE_STATE. If you print the status of the package P again using print_status, you see that cnt is 0 again and the cursor is CLOSED. If the call to DBMS_SESSION.MODIFY_PACKAGE_STATE had not been made, then the second print_status would have printed 111 and OPEN.

```
create or replace package P is
  cnt    number := 0;
  cursor c is select * from emp;
  procedure print_status;
end P;
/
show errors;

create or replace package body P is
  procedure print_status is
  begin
    dbms_output.put_line('P.cnt = ' || cnt);
    if c%ISOPEN then
      dbms_output.put_line('P.c is OPEN');
    else
      dbms_output.put_line('P.c is CLOSED');
    end if;
  end;
end P;
/
show errors;

SQL> set serveroutput on;
SQL> begin
  2   P.cnt := 111;
  3   open p.c;
  4   P.print_status;
  5  end;
  6  /
P.cnt = 111
P.c is OPEN

PL/SQL procedure successfully completed.

SQL> begin
  2   dbms_session.modify_package_state(dbms_session.reinitialize);
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL>
SQL> begin
  2   P.print_status;
```

```
  3  end;
  4  /
P.cnt = 0
P.c is CLOSED

PL/SQL procedure successfully completed.
```

UNIQUE_SESSION_ID Function

This function returns an identifier that is unique for all sessions currently connected to this database. Multiple calls to this function during the same session always return the same result.

Syntax

```
DBMS_SESSION.UNIQUE_SESSION_ID
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(unique_session_id,WNDS,RNDS,WNPS);
```

Returns

Table 65–12 UNIQUE_SESSION_ID Function Returns

Return	Description
`unique_session_id`	Returns up to 24 bytes.

IS_ROLE_ENABLED Function

This function determines if the named role is enabled for this session.

Syntax

```
DBMS_SESSION.IS_ROLE_ENABLED (
   rolename VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 65–13 IS_ROLE_ENABLED Function Parameters

Parameter	Description
rolename	Name of the role.

Returns

Table 65–14 IS_ROLE_ENABLED Function Returns

Return	Description
is_role_enabled	TRUE or FALSE, depending on whether the role is enabled.

IS_SESSION_ALIVE Function

This function determines if the specified session is active.

Syntax

```
DBMS_SESSION.IS_SESSION_ALIVE (
   uniqueid VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 65–15 IS_SESSION_ALIVE Function Parameters

Parameter	Description
uniqueid	Unique ID of the session: This is the same one as returned by UNIQUE_SESSION_ID.

Returns

Table 65–16 IS_SESSION_ALIVE Function Returns

Return	Description
is_session_alive	TRUE or FALSE, depending on whether the session is active.

SET_CLOSE_CACHED_OPEN_CURSORS Procedure

This procedure turns `close_cached_open_cursors` on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET CLOSE_CACHED_OPEN_CURSORS ...
```

Syntax

```
DBMS_SESSION.SET_CLOSE_CACHED_OPEN_CURSORS (
   close_cursors BOOLEAN);
```

Parameters

Table 65–17 SET_CLOSE_CACHED_OPEN_CURSORS Procedure Parameters

Parameter	Description
`close_cursors`	TRUE or FALSE

FREE_UNUSED_USER_MEMORY Procedure

This procedure reclaims unused memory after performing operations requiring large amounts of memory (more than 100K).

Examples of operations that use large amounts of memory include:

- Large sorting where entire `sort_area_size` is used and `sort_area_size` is hundreds of KB.
- Compiling large PL/SQL packages, procedures, or functions.
- Storing hundreds of KB of data within PL/SQL indexed tables.

You can monitor user memory by tracking the statistics "session uga memory" and "session pga memory" in the `v$sesstat` or `v$statname` fixed views. Monitoring these statistics also shows how much memory this procedure has freed.

Note: This procedure should only be used in cases where memory is at a premium. It should be used infrequently and judiciously.

Syntax

```
DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
```

Returns

The behavior of this procedure depends upon the configuration of the server operating on behalf of the client:

- **Dedicated server**: This returns unused PGA memory and session memory to the operating system. Session memory is allocated from the PGA in this configuration.
- **Shared server**: This returns unused session memory to the `shared_pool`. Session memory is allocated from the `shared_pool` in this configuration.

Usage Notes

In order to free memory using this procedure, the memory must not be in use.

After an operation allocates memory, only the same type of operation can reuse the allocated memory. For example, after memory is allocated for sort, even if the sort is complete and the memory is no longer in use, only another sort can reuse the sort-allocated memory. For both sort and compilation, after the operation is complete, the memory is no longer in use, and the user can call this procedure to free the unused memory.

An indexed table implicitly allocates memory to store values assigned to the indexed table's elements. Thus, the more elements in an indexed table, the more memory the RDBMS allocates to the indexed table. As long as there are elements within the indexed table, the memory associated with an indexed table is in use.

The scope of indexed tables determines how long their memory is in use. Indexed tables declared globally are indexed tables declared in packages or package bodies. They allocate memory from session memory. For an indexed table declared globally, the memory remains in use for the lifetime of a user's login (lifetime of a user's session), and is freed after the user disconnects from ORACLE.

Indexed tables declared locally are indexed tables declared within functions, procedures, or anonymous blocks. These indexed tables allocate memory from PGA memory. For an indexed table declared locally, the memory remains in use for as long as the user is still running the procedure, function, or anonymous block in which the indexed table is declared.After the procedure, function, or anonymous block is finished running, the memory is then available for other locally declared indexed tables to use (in other words, the memory is no longer in use).

Assigning an uninitialized, "empty" indexed table to an existing index table is a method to explicitly re-initialize the indexed table and the memory associated with the indexed table. After this operation, the memory associated with the indexed table is no longer in use, making it available to be freed by calling this procedure. This method is particularly useful on indexed tables declared globally which can grow during the lifetime of a user's session, as long as the user no longer needs the contents of the indexed table.

The memory rules associated with an indexed table's scope still apply; this method and this procedure, however, allow users to intervene and to explicitly free the memory associated with an indexed table.

Example

The following PL/SQL illustrates the method and the use of procedure FREE_UNUSED_USER_MEMORY.

```
CREATE PACKAGE foobar
   type number_idx_tbl is table of number indexed by binary_integer;

   store1_table  number_idx_tbl;     --  PL/SQL indexed table
   store2_table  number_idx_tbl;     --  PL/SQL indexed table
   store3_table  number_idx_tbl;     --  PL/SQL indexed table
   ...
END;            --  end of foobar

DECLARE
   ...
   empty_table   number_idx_tbl;     --  uninitialized ("empty") version
BEGIN
   FOR i in 1..1000000 loop
     store1_table(i) := i;           --  load data
   END LOOP;
   ...
   store1_table := empty_table;      --  "truncate" the indexed table
   ...
   -
   dbms_session.free_unused_user_memory;  -- give memory back to system

   store1_table(1) := 100;           --  index tables still declared;
   store2_table(2) := 200;           --  but truncated.
   ...
END;
```

SET_CONTEXT Procedure

This procedure sets or resets the value of a context attribute.

Syntax

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
   value     VARCHAR2,
   username  VARCHAR2,
   client_id VARCHAR2);
```

Parameters

Table 65–18 SET_CONTEXT Procedure Parameters

Parameter	Description
namespace	Name of the namespace to use for the application context (limited to 30 bytes).
attribute	Name of the attribute to be set (limited to 30 bytes).
value	Value to be set (limited to 4 kilobytes).
username	The username attribute of the application context
client_id	The application-specific identifier of the current database session.

Usage Notes

The caller of this function must be in the calling stack of a procedure which has been associated to the context namespace through a CREATE CONTEXT statement. The checking of the calling stack does not cross DBMS boundary.

There is no limit on the number of attributes that can be set in a namespace. An attribute value remains for user session, or until it is reset by the user.

LIST_CONTEXT Procedure

This procedure returns a list of active namespaces and contexts for the current session.

Syntax

```
TYPE AppCtxRecTyp IS RECORD (
   namespace VARCHAR2(30),
   attribute VARCHAR2(30),
   value     VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT (
   list OUT AppCtxTabTyp,
   size OUT NUMBER);
```

Parameters

Table 65–19 LIST_CONTEXT Procedure Parameters

Parameter	Description
list	Buffer to store a list of application context set in the current session.

Returns

Table 65–20 LIST_CONTEXT Procedure Returns

Return	Description
list	A list of (namespace, attribute, values) set in current session
size	Returns the number of entries in the buffer returned

Usage Notes

The context information in the list appears as a series of <namespace> <attribute> <value>. Because list is a table type variable, its size is dynamically adjusted to the size of returned list.

SWITCH_CURRENT_CONSUMER_GROUP Procedure

This procedure changes the current resource consumer group of a user's current session.

This lets you switch to a consumer group if you have the switch privilege for that particular group. If the caller is another procedure, then this enables the user to

switch to a consumer group for which the owner of that procedure has switch privilege.

Syntax

```
DBMS_SESSION.switch_current_consumer_group (
   new_consumer_group     IN  VARCHAR2,
   old_consumer_group     OUT VARCHAR2,
   initial_group_on_error IN  BOOLEAN);
```

Parameters

Table 65–21 SWITCH_CURRENT_CONSUMER_GROUP Procedure Parameters

Parameter	Description
new_consumer_group	Name of consumer group to which you want to switch.
old_consumer_group	Name of the consumer group from which you just switched out.
initial_group_on_error	If TRUE, then sets the current consumer group of the caller to his / her initial consumer group in the event of an error.

Returns

This procedure outputs the old consumer group of the user in the parameter old_consumer_group.

> **Note:** You can switch back to the old consumer group later using the value returned in old_consumer_group.

Exceptions

Table 65–22 SWITCH_CURRENT_CONSUMER_GROUP Procedure Exceptions

Exception	Description
29368	Non-existent consumer group.
1031	Insufficient privileges.
29396	Cannot switch to OTHER_GROUPS consumer group.

Usage Notes

The owner of a procedure must have privileges on the group from which a user was switched (old_consumer_group) in order to switch them back. There is one exception: The procedure can always switch the user back to his/her initial consumer group (skipping the privilege check).

By setting initial_group_on_error to TRUE, SWITCH_CURRENT_CONSUMER_GROUP puts the current session into the default group, if it can't put it into the group designated by new_consumer_group. The error associated with the attempt to move a session into new_consumer_group is raised, even though the current consumer group has been changed to the initial consumer group.

Example

```
CREATE OR REPLACE PROCEDURE high_priority_task is
   old_group varchar2(30);
   prev_group varchar2(30);
   curr_user varchar2(30);
BEGIN
  -- switch invoker to privileged consumer group. If we fail to do so, an
  -- error will be thrown, but the consumer group will not change
  -- because 'initial_group_on_error' is set to FALSE

  dbms_session.switch_current_consumer_group('tkrogrp1', old_group, FALSE);
  -- set up exception handler (in the event of an error, we do not want to
  -- return to caller while leaving the session still in the privileged
  -- group)

  BEGIN
    -- perform some operations while under privileged group

  EXCEPTION
    WHEN OTHERS THEN
     -- It is possible that the procedure owner does not have privileges
     -- on old_group. 'initial_group_on_error' is set to TRUE to make sure
     -- that the user is moved out of the privileged group in such a
     -- situation

     dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
     RAISE;
    END;

  -- we've succeeded. Now switch to old_group, or if cannot do so, switch
  -- to caller's initial consumer group
```

```
  dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
END high_priority_task;
/
```

66

DBMS_SHARED_POOL

DBMS_SHARED_POOL provides access to the shared pool, which is the shared memory area where cursors and PL/SQL objects are stored. DBMS_SHARED_POOL enables you to display the sizes of objects in the shared pool, and mark them for keeping or unkeeping in order to reduce memory fragmentation.

This chapter discusses the following topics:

Installation Notes

To create DBMS_SHARED_POOL, run the DBMSPOOL.SQL script. The PRVTPOOL.PLB script is automatically executed after DBMSPOOL.SQL runs. These scripts are *not* run by CATPROC.SQL.

Usage Notes

The procedures provided here may be useful when loading large PL/SQL objects. When large PL/SQL objects are loaded, users response time is affected because of the large number of smaller objects that need to be aged out from the shared pool to make room (due to memory fragmentation). In some cases, there may be insufficient memory to load the large objects.

DBMS_SHARED_POOL is also useful for frequently executed triggers. You may want to keep compiled triggers on frequently used tables in the shared pool. Additionally, DBMS_SHARED_POOL supports sequences. Sequence numbers are lost when a sequence is aged out of the shared pool. DBMS_SHARED_POOL is useful for keeping sequences in the shared pool and thus preventing the loss of sequence numbers.

Summary of DBMS_SHARED_POOL Subprograms

Table 66–1 DBMS_SHARED_POOL Subprograms

Subprogram	Description
SIZES Procedure on page 66-3	Shows objects in the shared pool that are larger than the specified size
KEEP Procedure on page 66-3	Keeps an object in the shared pool
UNKEEP Procedure on page 66-5	Unkeeps the named object
ABORTED_REQUEST_THRESHOLD Procedure on page 66-5	Sets the aborted request threshold for the shared pool

SIZES Procedure

This procedure shows objects in the shared_pool that are larger than the specified size. The name of the object is also given, which can be used as an argument to either the KEEP or UNKEEP calls.

Syntax

```
DBMS_SHARED_POOL.SIZES (
   minsize NUMBER);
```

Parameters

Table 66–2 SIZES Procedure Parameters

Parameter	Description
minsize	Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.

Usage Notes

Issue the SQLDBA or SQLPLUS 'SET SERVEROUTPUT ON SIZE XXXXX' command prior to using this procedure so that the results are displayed.

KEEP Procedure

This procedure keeps an object in the shared pool. Once an object has been kept in the shared pool, it is not subject to aging out of the pool. This may be useful for frequently used large objects. When large objects are brought into the shared pool, several objects may need to be aged out to create a contiguous area large enough.

Syntax

```
DBMS_SHARED_POOL.KEEP (
   name VARCHAR2,
   flag CHAR      DEFAULT 'P');
```

> **Note:** This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 66–3 KEEP Procedure Parameters

Parameter	Description
name	Name of the object to keep. The value for this identifier is the concatenation of the address and hash_value columns from the v$sqlarea view. This is displayed by the SIZES procedure. Currently, TABLE and VIEW objects may not be kept.
flag	(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package / procedure / function and resolves the name. Set to 'P' or 'p' to fully specify that the input is the name of a package / procedure / function. Set to 'T' or 't' to specify that the input is the name of a type. Set to 'R' or 'r' to specify that the input is the name of a trigger. Set to 'Q' or 'q' to specify that the input is the name of a sequence. In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.

Exceptions

An exception is raised if the named object cannot be found.

Usage Notes

There are two kinds of objects:

- PL / SQL objects, triggers, sequences, and types which are specified by name
- SQL cursor objects which are specified by a two-part number (indicating a location in the shared pool).

For example:

```
DBMS_SHARED_POOL.KEEP('scott.hispackage')
```

This keeps package HISPACKAGE, owned by SCOTT. The names for PL / SQL objects follow SQL rules for naming objects (for example, delimited identifiers and multibyte names are allowed). A cursor can be kept by DBMS_SHARED_

POOL.KEEP('0034CDFF, 20348871'). The complete hexadecimal address must be in the first 8 characters.

UNKEEP Procedure

This procedure unkeeps the named object.

Syntax

```
DBMS_SHARED_POOL.UNKEEP (
   name VARCHAR2,
   flag CHAR     DEFAULT 'P');
```

> **Caution:** This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 66–4 UNKEEP Procedure Parameters

Parameter	Description
name	Name of the object to unkeep. See description of the name object for the KEEP procedure.
flag	See description of the flag parameter for the KEEP procedure.

Exceptions

An exception is raised if the named object cannot be found.

ABORTED_REQUEST_THRESHOLD Procedure

This procedure sets the aborted request threshold for the shared pool.

Syntax

```
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (
   threshold_size NUMBER);
```

Parameters

Table 66–5 ABORTED_REQUEST_THRESHOLD Procedure Parameters

Parameter	Description
`threshold_size`	Size, in bytes, of a request which does not try to free unpinned (not "unkeep-ed") memory within the shared pool. The range of `threshold_size` is 5000 to ~2 GB inclusive.

Exceptions

An exception is raised if the threshold is not in the valid range.

Usage Notes

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an 'ALTER SYSTEM FLUSH SHARED_POOL'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than `thresh_hold` size. This user gets the 'out of memory' error without attempting to search the LRU list.

67

DBMS_SPACE

The DBMS_SPACE package enables you to analyze segment growth and space requirements.

This chapter discusses the following topics:

- Security
- Requirements
- Summary of DBMS_SPACE Subprograms

Security

This package runs with SYS privileges.

Requirements

The execution privilege is granted to PUBLIC. Subprograms in this package run under the caller security. The user must have ANALYZE privilege on the object.

Summary of DBMS_SPACE Subprograms

Table 67–1 DBMS_SPACE Subprograms

Subprogram	Description
UNUSED_SPACE Procedure on page 67-2	Returns information about unused space in an object (table, index, or cluster).
FREE_BLOCKS Procedure on page 67-3	Returns information about free blocks in an object (table, index, or cluster).
SPACE_USAGE Procedure on page 67-5	Returns information about free blocks in a bitmapped segment.

UNUSED_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```
DBMS_SPACE.UNUSED_SPACE (
   segment_owner              IN  VARCHAR2,
   segment_name               IN  VARCHAR2,
   segment_type               IN  VARCHAR2,
   total_blocks               OUT NUMBER,
   total_bytes                OUT NUMBER,
   unused_blocks              OUT NUMBER,
   unused_bytes               OUT NUMBER,
   last_used_extent_file_id   OUT NUMBER,
   last_used_extent_block_id  OUT NUMBER,
   last_used_block            OUT NUMBER,
   partition_name             IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 67–2 UNUSED_SPACE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed.
segment_name	Segment name of the segment to be analyzed.
segment_type	Type of the segment to be analyzed: TABLE TABLE PARTITION TABLE SUBPARTITION INDEX INDEX PARTITION INDEX SUBPARTITION CLUSTER LOB
total_blocks	Returns total number of blocks in the segment.
total_bytes	Returns total number of blocks in the segment, in bytes.
unused_blocks	Returns number of blocks which are not used.
unused_bytes	Returns, in bytes, number of blocks which are not used.
last_used_extent_file_id	Returns the file ID of the last extent which contains data.
last_used_extent_block_id	Returns the block ID of the last extent which contains data.
last_used_block	Returns the last block within this extent which contains data.
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

FREE_BLOCKS Procedure

This procedure returns information about free blocks in an object (table, index, or cluster). See "SPACE_USAGE Procedure" for returning free block information in a bitmapped segment.

Syntax

```
DBMS_SPACE.FREE_BLOCKS (
   segment_owner     IN  VARCHAR2,
   segment_name      IN  VARCHAR2,
   segment_type      IN  VARCHAR2,
   freelist_group_id IN  NUMBER,
   free_blks         OUT NUMBER,
   scan_limit        IN  NUMBER DEFAULT NULL,
   partition_name    IN  VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 67–3 FREE_BLOCKS Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed.
segment_name	Segment name of the segment to be analyzed.
segment_type	Type of the segment to be analyzed: TABLE TABLE PARTITION TABLE SUBPARTITION INDEX INDEX PARTITION INDEX SUBPARTITION CLUSTER LOB
freelist_group_id	Freelist group (instance) whose free list size is to be computed.
free_blks	Returns count of free blocks for the specified group.
scan_limit	Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, "Do I have X blocks on the free list?"

Table 67–3 FREE_BLOCKS Procedure Parameters

Parameter	Description
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

Example 1

The following declares the necessary bind variables and executes.

```
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks,
   :total_bytes,:unused_blocks, :unused_bytes, :lastextf,
   :last_extb, :lastusedblock);
```

This fills the unused space information for bind variables in EMP table in SCOTT schema.

Example 2

The following uses the CLUS cluster in SCOTT schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in CLUS.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

Note: An error is raised if scan_limit is not a positive number.

SPACE_USAGE Procedure

This procedure shows the space usage of data blocks under the segment High Water Mark. The bitmap blocks, segment header, and extent map blocks are not accounted for by this procedure. This procedure can only be used on tablespaces that are created with auto segment space management.

Syntax

```
DBMS_SPACE.SPACE_USAGE(
   segment_owner IN varchar2,
   segment_name IN varchar2,
   segment_type IN varchar2,
   unformatted_blocks OUT number,
   unformatted_bytes OUT number,
```

```
fs1_blocks OUT number,
fs1_bytes  OUT number,
fs2_blocks OUT number,
fs2_bytes  OUT number,
fs3_blocks OUT number,
fs3_bytes  OUT number,
fs4_blocks OUT number,
fs4_bytes  OUT number,
full_blocks OUT number,
full_bytes OUT number,
partition_name IN varchar2 DEFAULT NULL);
```

Parameters

Table 67–4 SPACE_USAGE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Name of the segment to be analyzed
partition_name	Partition name of the segment to be analyzed
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER)
OUTPUT ARGUMENTS	
unformatted_blocks	Total number of blocks that are unformatted
unformatted bytes	Total number of bytes that are unformatted
fs1_blocks	Number of blocks that has at least 0 to 25% free space
fs1_bytes	Number of bytes that has at least 0 to 25% free space
fs2_blocks	Number of blocks that has at least 25 to 50% free space
fs2_bytes	Number of bytes that has at least 25 to 50% free space
fs3_blocks	Number of blocks that has at least 50 to 75% free space
fs3_bytes	Number of bytes that has at least 50 to 75% free space
fs4_blocks	Number of blocks that has at least 75 to 100% free space
fs4_bytes	Number of bytes that has at least 75 to 100% free space
ful1_blocks	Total number of blocks that are full in the segment
full_bytes	Total number of bytes that are full in the segment

Example

```
variable unf number;
variable unfb number;
variable fs1 number;
variable fs1b number;
variable fs2 number;
variable fs2b number;
variable fs3 number;
variable fs3b number;
variable fs4 number;
variable fs4b number;
variable full number;
variable fullb number;

begin
dbms_space.space_usage('U1','T',
                       'TABLE',
                       :unf, :unfb,
                       :fs1, :fs1b,
                       :fs2, :fs2b,
                       :fs3, :fs3b,
                       :fs4, :fs4b,
                       :full, :fullb);
end;
/
print unf ;
print unfb ;
print fs4 ;
print fs4b;
print fs3 ;
print fs3b;
print fs2 ;
print fs2b;
print fs1 ;
print fs1b;
print full;
print fullb;
```

68

DBMS_SPACE_ADMIN

The DBMS_SPACE_ADMIN package provides functionality for locally managed tablespaces.

> **See Also:** *Oracle9i Database Administrator's Guide* for an example and description of using DBMS_SPACE_ADMIN.

This chapter discusses the following topics:

Security

This package runs with SYS privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

SYSTEM Tablespace Migration: Conditions

Before you migrate the SYSTEM tablespace, you should migrate any dictionary-managed tablespaces that you may want to use in read / write mode to locally managed. After the SYSTEM tablespace is migrated, you cannot change dictionary-managed tablespaces to read / write.

See Also:

- *Oracle9i Database Administrator's Guide*
- "TABLESPACE_MIGRATE_TO_LOCAL Procedure" on page 68-11

Before migrating the SYSTEM tablespace, the following conditions must be met. These conditions are enforced by the TABLESPACE_MIGRATE_TO_LOCAL procedure, except for the cold backup.

- The database must have a default temporary tablespace that is not SYSTEM.
- Dictionary-managed tablespaces cannot have any rollback segments.
- A locally managed tablespace must have at least one online rollback segment. If you are using automatic undo management, an undo tablespace must be online.
- All tablespaces—except the tablespace containing the rollback segment or the undo tablespace—must be read-only.
- You must have a cold backup of the database.
- The system must be in restricted mode.

Constants for DBMS_SPACE_ADMIN Constants

Table 68–1 DBMS_SPACE_ADMIN Constants

Constant	Description
SEGMENT_VERIFY_EXTENTS	Verifies that the space owned by segment is appropriately reflected in the bitmap as used.

Table 68–1 DBMS_SPACE_ADMIN Constants

Constant	Description
SEGMENT_VERIFY_EXTENTS_GLOBAL	Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it.
SEGMENT_MARK_CORRUPT	Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation).
SEGMENT_MARK_VALID	Marks a corrupt temporary segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.
SEGMENT_DUMP_EXTENT_MAP	Dumps the extent map for a given segment.
TABLESPACE_VERIFY_BITMAP	Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent.
TABLESPACE_EXTENT_MAKE_FREE	Makes this range (extent) of space free in the bitmaps.
TABLESPACE_EXTENT_MAKE_USED	Makes this range (extent) of space used in the bitmaps.

Summary of DBMS_SPACE_ADMIN Subprograms

Table 68–2 DBMS_SPACE_ADMIN Subprograms

Subprogram	Description
SEGMENT_VERIFY Procedure on page 68-4	Verifies the consistency of the extent map of the segment.
SEGMENT_CORRUPT Procedure on page 68-5	Marks the segment corrupt or valid so that appropriate error recovery can be done.
SEGMENT_DROP_CORRUPT Procedure on page 68-6	Drops a segment currently marked corrupt (without reclaiming space).
SEGMENT_DUMP Procedure on page 68-7	Dumps the segment header and extent maps of a given segment.

Table 68–2 DBMS_SPACE_ADMIN Subprograms

Subprogram	Description
TABLESPACE_VERIFY Procedure on page 68-8	Verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.
TABLESPACE_FIX_BITMAPS Procedure on page 68-8	Marks the appropriate DBA range (extent) as free or used in bitmap.
TABLESPACE_REBUILD_ BITMAPS Procedure on page 68-9	Rebuilds the appropriate bitmaps.
TABLESPACE_REBUILD_ QUOTAS Procedure on page 68-10	Rebuilds quotas for given tablespace.
TABLESPACE_MIGRATE_ FROM_LOCAL Procedure on page 68-11	Migrates a locally-managed tablespace to dictionary-managed tablespace.
TABLESPACE_MIGRATE_TO_ LOCAL Procedure on page 68-11	Migrates a tablespace from dictionary managed format to locally managed format.
TABLESPACE_RELOCATE_ BITMAPS Procedure on page 68-13	Relocates the bitmaps to the destination specified.
TABLESPACE_FIX_SEGMENT_ STATES Procedure on page 68-14	Fixes the state of the segments in a tablespace in which migration was aborted.

SEGMENT_VERIFY Procedure

This procedure verifies that the extent map of the segment is consistent with the bitmap.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   verify_option           IN    POSITIVE  DEFAULT SEGMENT_VERIFY_EXTENTS);
```

Parameters

Table 68–3 SEGMENT_VERIFY Procedure Parameters

Parameters	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
verify_option	What kind of check to do: SEGMENT_VERIFY_EXTENTS or SEGMENT_VERIFY_EXTENTS_GLOBAL.

Usage Notes

Anomalies are output as dba-range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all dba-ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not considered free, used space considered free, and the same space considered used by multiple segments.

Example

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps in sync.

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33, 1);
```

> **Note:** All DBMS_SPACE_ADMIN package examples use the tablespace USERS which contains SCOTT.EMP.

SEGMENT_CORRUPT Procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be done. It cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
```

```
   corrupt_option          IN    POSITIVE  DEFAULT SEGMENT_MARK_CORRUPT);
```

Parameters

Table 68–4 SEGMENT_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
corrupt_option	SEGMENT_MARK_CORRUPT (default) or SEGMENT_MARK_VALID.

Example

The following example marks the segment as corrupt:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 3);
```

Alternately, the next example marks a corrupt segment valid:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 4);
```

SEGMENT_DROP_CORRUPT Procedure

This procedure drops a segment currently marked corrupt (without reclaiming space). For this to work, the segment should have been marked *temporary*. To mark a corrupt segment as temporary, issue a DROP command on the segment.

The procedure cannot be used on the SYSTEM tablespace.

The space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS Procedure or the TABLESPACE_REBUILD_BITMAPS Procedure.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE);
```

Parameters

Table 68–5 SEGMENT_DROP_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.

Example

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);
```

SEGMENT_DUMP Procedure

This procedure dumps the segment header and extent map blocks of the given segment.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   dump_option             IN    POSITIVE  DEFAULT SEGMENT_DUMP_EXTENT_MAP);
```

Parameters

Table 68–6 SEGMENT_DUMP Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
dump_option	SEGMENT_DUMP_EXTENT_MAP

Example

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);
```

TABLESPACE_VERIFY Procedure

This procedure verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (
   tablespace_name         IN    VARCHAR2,
   verify_option           IN    POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

Table 68–7 TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
verify_option	TABLESPACE_VERIFY_BITMAP

Example

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('USERS');
```

TABLESPACE_FIX_BITMAPS Procedure

This procedure marks the appropriate DBA range (extent) as free or used in bitmap. It cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   dbarange_relative_file  IN    POSITIVE,
   dbarange_begin_block    IN    POSITIVE,
   dbarange_end_block      IN    POSITIVE,
   fix_option              IN    POSITIVE);
```

Parameters

Table 68–8 TABLESPACE_FIX_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
dbarange_relative_file	Relative file number of DBA range (extent).
dbarange_begin_block	Block number of beginning of extent.
dbarange_end_block	Block number (inclusive) of end of extent.
fix_option	TABLESPACE_EXTENT_MAKE_FREE or TABLESPACE_EXTENT_MAKE_USED.

Example

The following example marks bits for 50 blocks for relative file number 4, beginning at block number 33 and ending at 83, as USED in bitmaps.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83, 7);
```

Alternately, specifying an option of 8 marks the bits FREE in bitmaps. The BEGIN and END blocks should be in extent boundary and should be extent multiple. Otherwise, an error is raised.

TABLESPACE_REBUILD_BITMAPS Procedure

This procedure rebuilds the appropriate bitmaps. If no bitmap block DBA is specified, then it rebuilds all bitmaps for the given tablespace.

The procedure cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   bitmap_relative_file    IN    POSITIVE   DEFAULT NULL,
   bitmap_block            IN    POSITIVE   DEFAULT NULL);
```

Parameters

Table 68–9 TABLESPACE_REBUILD_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
bitmap_relative_file	Relative file number of bitmap block to rebuild.
bitmap_block	Block number of bitmap block to rebuild.

Example

The following example rebuilds bitmaps for all the files in the USERS tablespace.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

Note: Only full rebuild is supported.

TABLESPACE_REBUILD_QUOTAS Procedure

This procedure rebuilds quotas for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (
   tablespace_name         IN    VARCHAR2);
```

Parameters

Table 68–10 TABLESPACE_REBUILD_QUOTAS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace

Example

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('USERS');
```

TABLESPACE_MIGRATE_FROM_LOCAL Procedure

This procedure migrates a locally-managed tablespace to a dictionary-managed tablespace. You cannot use this procedure for SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
   tablespace_name          IN     VARCHAR2);
```

Parameter

Table 68–11 TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameter

Parameter	Description
tablespace_name	Name of tablespace

Usage Notes

The tablespace must be kept online and read/write during migration. Migration of temporary tablespaces and migration of SYSTEM tablespaces are not supported.

Example

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');
```

TABLESPACE_MIGRATE_TO_LOCAL Procedure

Use this procedure to migrate the tablespace from a dictionary-managed format to a locally managed format. Tablespaces migrated to locally managed format are user managed.

Caution: Do not migrate the SYSTEM tablespace without a clear understanding of the conditions that must be met. Refer to "SYSTEM Tablespace Migration: Conditions" on page 68-2.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL (
   tablespace_name
   allocation_unit
   relative_fno)
```

Parameters

Table 68–12 Parameters for TABLESPACE_MIGRATE_TO_LOCAL

Parameter Name	Purpose	Datatype	Parameter Type
tablespace_name	Name of the tablespace to be migrated.	VARCHAR	IN
allocation_unit	Unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace.	INTEGER	IN
relative_fno	Relative File Number of the file where the bitmap blocks should be placed (optional)	INTEGER	IN

Usage Notes

The tablespace must be kept online and read / write during migration. Note that temporary tablespaces cannot be migrated.

Allocation Unit may be specified optionally. The default is calculated by the system based on the highest common divisor of all extents (used or free) for the tablespace. This number is further trimmed based on the `MINIMUM EXTENT` for the tablespace (5 if `MINIMUM EXTENT`T is not specified). Thus, the calculated value will not be larger than the `MINIMUM EXTENT` for the tablespace. The last free extent in every file will be ignored for GCD calculation. If you specify the unit size, it has to be a factor of the UNIT size calculated by the system, otherwise an error message is returned.

The Relative File Number parameter is used to place the bitmaps in a desired file. If space is not found in the file, an error is issued. The datafile specified should be part of the tablespace being migrated. If the datafile is not specified then the system will choose a datafile in which to place the initial bitmap blocks. If space is not found for the initial bitmaps, an error will be raised.

Example

To migrate a tablespace 'TS1' with minimum extent size 1m, use

```
execute dbms_space_admin.tablespace_migrate_to_local('TS1', 512, 2);
```

The bitmaps will be placed in file with relative file number 2.

TABLESPACE_RELOCATE_BITMAPS Procedure

Use this procedure to relocate the bitmaps to the destination specified. Migration of a tablespace from dictionary managed to locally managed format could result in the creation of SPACE HEADER segment that contains the bitmap blocks. The SPACE HEADER segment is treated as user data. If the user wishes to explicitly resize a file at or below the space header segment, an error is issued. Use the `tablespace_relocate_bitmaps` command to move the control information to a different destination and then resize the file.

This procedure cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS (
   tablespace_name
   relative_fno
   block_number )
```

Parameters

Table 68–13 Parameters for TABLESPACE_RELOCATE_BITMAPS

Parameter Name	Purpose	Datatype	Parameter Type
tablespace_name	Name of Tablespace.	VARCHAR	IN
relative_fno	Relative File Number of the destination file.	NUMBER	IN
block_number	Block Number of the destination dba.	NUMBER	IN

Usage Notes

The tablespace must be kept online and read / write during relocation of bitmaps. Can be done only on migrated locally managed tablespaces.

Example

```
execute  dbms_space_admin.tablespace_relocate_bitmaps('TS1', 3, 4);
```

Moves the bitmaps to file 3, block 4.

> **Note:** The source and the destination addresses should not overlap. The destination block number is rounded down to the unit boundary. If there is user data in that location an error is raised.

TABLESPACE_FIX_SEGMENT_STATES Procedure

Use this procedure to fix the state of the segments in a tablespace in which migration was aborted. During tablespace migration to or from local, the segments are put in a transient state. If migration is aborted, the segment states are corrected by SMON when event 10906 is set. Database with segments is such a transient state cannot be downgraded. The procedure can be used to fix the state of such segments.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES (
   tablespace_name);
```

Parameters

Table 68–14 Parameter for TABLESPACE_FIX_SEGMENT_STATES

Parameter Name	Purpose	Datatype	Parameter Type
tablespace_name	Name of the tablespace whose segments need to be fixed.	VARCHAR	IN

Usage Notes

The tablespace must be kept online and read / write when this procedure is called.

Example

```
execute dbms_space_admin.tablespace_fix_segment_states('TS1');
```

69

DBMS_SQL

Oracle lets you to write stored procedures and anonymous PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program; rather, they are stored in character strings that are input to, or built by, the program at runtime. This enables you to create more general-purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

Additionally, DBMS_SQL enables you to parse any data manipulation language (DML) or data definition language (DDL) statement. Therefore, you can parse DDL statements directly using PL/SQL. For example, you might now choose to enter a DROP TABLE statement from within a stored procedure by using the PARSE procedure supplied with the DBMS_SQL package.

Note: Oracle8*i* introduces native dynamic SQL, an alternative to DBMS_SQL. Using native dynamic SQL, you can place dynamic SQL statements directly into PL/SQL blocks.

In most situations, native dynamic SQL can replace DBMS_SQL. Native dynamic SQL is easier to use and performs better than DBMS_SQL.

See Also: For more information on native dynamic SQL, see *PL/SQL User's Guide and Reference.*

For a comparison of DBMS_SQL and native dynamic SQL, see *Oracle9i Application Developer's Guide - Fundamentals.*

This chapter discusses the following topics:

- Using DBMS_SQL
- Constants, Types, and Exceptions for DBMS_SQL
- Security
- Processing Queries
- Examples
- Processing Updates, Inserts, and Deletes
- Locating Errors
- Summary of DBMS_SQL Subprograms

Using DBMS_SQL

The ability to use dynamic SQL from within stored procedures generally follows the model of the Oracle Call Interface (OCI).

See Also: *Oracle Call Interface Programmer's Guide*

PL/SQL differs somewhat from other common programming languages, such as C. For example, addresses (also called pointers) are not user-visible in PL/SQL. As a result, there are some differences between the Oracle Call Interface and the DBMS_SQL package. These differences include the following:

- The OCI uses bind by address, while the DBMS_SQL package uses bind by value.
- With DBMS_SQL you must call VARIABLE_VALUE to retrieve the value of an OUT parameter for an anonymous block, and you must call COLUMN_VALUE after fetching rows to actually retrieve the values of the columns in the rows into your program.
- The current release of the DBMS_SQL package does not provide CANCEL cursor procedures.
- Indicator variables are not required, because NULLs are fully supported as values of a PL/SQL variable.

A sample usage of the DBMS_SQL package follows. For users of the Oracle Call Interfaces, this code should seem fairly straightforward.

Example

This example does not actually require the use of dynamic SQL, because the text of the statement is known at compile time. It does, however, illustrate the concepts of this package.

The DEMO procedure deletes all of the employees from the EMP table whose salaries are greater than the salary that you specify when you run DEMO.

```
CREATE OR REPLACE PROCEDURE demo(salary IN NUMBER) AS
    cursor_name INTEGER;
    rows_processed INTEGER;
BEGIN
    cursor_name := dbms_sql.open_cursor;
    DBMS_SQL.PARSE(cursor_name, 'DELETE FROM emp WHERE sal > :x',
                   dbms_sql.native);
    DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', salary);
```

```
    rows_processed := dbms_sql.execute(cursor_name);
    DBMS_SQL.close_cursor(cursor_name);
EXCEPTION
WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Constants, Types, and Exceptions for DBMS_SQL

Constants

```
v6 constant INTEGER     := 0;
native constant INTEGER := 1;
v7 constant INTEGER     := 2;
```

Types

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
TYPE desc_rec  IS RECORD (
      col_type            BINARY_INTEGER := 0,
      col_max_len         BINARY_INTEGER := 0,
      col_name            VARCHAR2(32)   := '',
      col_name_len        BINARY_INTEGER := 0,
      col_schema_name     VARCHAR2(32)   := '',
      col_schema_name_len BINARY_INTEGER := 0,
      col_precision       BINARY_INTEGER := 0,
      col_scale           BINARY_INTEGER := 0,
      col_charsetid       BINARY_INTEGER := 0,
      col_charsetform     BINARY_INTEGER := 0,
      col_null_ok         BOOLEAN        := TRUE);
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

Bulk SQL Types

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
type Urowid_Table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
```

Exceptions

```
inconsistent_type exception;
```

```
pragma exception_init(inconsistent_type, -6562);
```

This exception is raised by procedure COLUMN_VALUE or VARIABLE_VALUE when the type of the given OUT parameter (for where to put the requested value) is different from the type of the value.

Execution Flow

OPEN_CURSOR

To process a SQL statement, you must have an open cursor. When you call the OPEN_CURSOR function, you receive a cursor ID number for the data structure representing a valid cursor maintained by Oracle. These cursors are distinct from cursors defined at the precompiler, OCI, or PL/SQL level, and are used only by the DBMS_SQL package.

PARSE

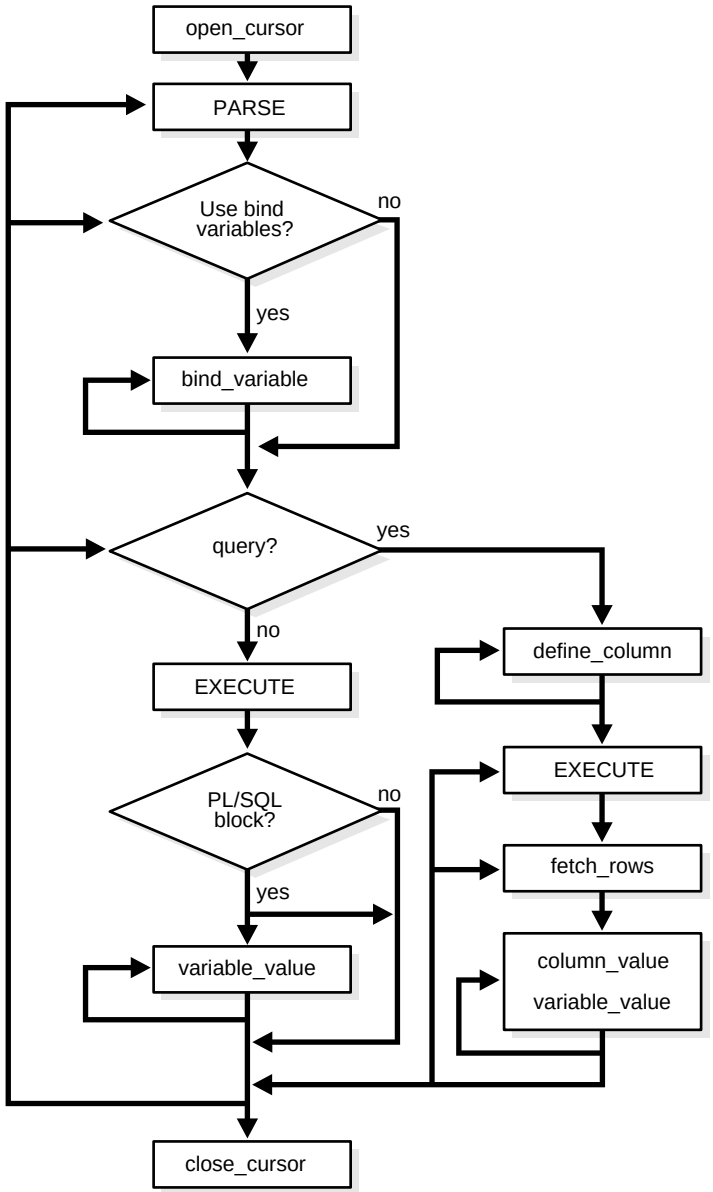
Every SQL statement must be parsed by calling the PARSE procedure. Parsing the statement checks the statement's syntax and associates it with the cursor in your program.

You can parse any DML or DDL statement. DDL statements are run on the parse, which performs the implied commit.

Note: When parsing a DDL statement to drop a package or a procedure, a deadlock can occur if you're still using a procedure in the package. After a call to a procedure, that procedure is considered to be in use until execution has returned to the user side. Any such deadlock timeouts after five minutes.

The execution flow of DBMS_SQL is shown in Figure 69–1.

Figure 69–1 DBMS_SQL Execution Flow



BIND_VARIABLE or BIND_ARRAY

Many DML statements require that data in your program be input to Oracle. When you define a SQL statement that contains input data to be supplied at runtime, you must use placeholders in the SQL statement to mark where data must be supplied.

For each placeholder in the SQL statement, you must call one of the bind procedures, BIND_VARIABLE or BIND_ARRAY, to supply the value of a variable in your program (or the values of an array) to the placeholder. When the SQL statement is subsequently run, Oracle uses the data that your program has placed in the output and input, or bind, variables.

DBMS_SQL can run a DML statement multiple times — each time with a different bind variable. The BIND_ARRAY procedure lets you bind a collection of scalars, each value of which is used as an input variable once for each EXECUTE. This is similar to the array interface supported by the OCI.

DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY

The columns of the row being selected in a SELECT statement are identified by their relative positions as they appear in the select list, from left to right. For a query, you must call one of the define procedures (DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY) to specify the variables that are to receive the SELECT values, much the way an INTO clause does for a static query.

Use the DEFINE_COLUMN_LONG procedure to define LONG columns, in the same way that DEFINE_COLUMN is used to define non-LONG columns. You must call DEFINE_COLUMN_LONG before using the COLUMN_VALUE_LONG procedure to fetch from the LONG column.

Use the DEFINE_ARRAY procedure to define a PL/SQL collection into which you want to fetch rows in a single SELECT statement. DEFINE_ARRAY provides an interface to fetch multiple rows at one fetch. You must call DEFINE_ARRAY before using the COLUMN_VALUE procedure to fetch the rows.

EXECUTE

Call the EXECUTE function to run your SQL statement.

FETCH_ROWS or EXECUTE_AND_FETCH

The FETCH_ROWS function retrieves the rows that satisfy the query. Each successive fetch retrieves another set of rows, until the fetch is unable to retrieve anymore rows. Instead of calling EXECUTE and then FETCH_ROWS, you may find it more efficient to call EXECUTE_AND_FETCH if you are calling EXECUTE for a single execution.

VARIABLE_VALUE, COLUMN_VALUE, or COLUMN_VALUE_LONG

For queries, call COLUMN_VALUE to determine the value of a column retrieved by the FETCH_ROWS call. For anonymous blocks containing calls to PL/SQL procedures or DML statements with returning clause, call VARIABLE_VALUE to retrieve the values assigned to the output variables when statements were run.

To fetch just part of a LONG database column (which can be up to two gigabytes in size), use the COLUMN_VALUE_LONG procedure. You can specify the offset (in bytes) into the column value, and the number of bytes to fetch.

CLOSE_CURSOR

When you no longer need a cursor for a session, close the cursor by calling CLOSE_CURSOR. If you are using an Oracle Open Gateway, then you may need to close cursors at other times as well. Consult your *Oracle Open Gateway* documentation for additional information.

If you neglect to close a cursor, then the memory used by that cursor remains allocated even though it is no longer needed.

Security

Definer Rights Modules

Definer rights modules run under the privileges of the owner of the module. DBMS_SQL subprograms called from definer rights modules run with respect to the schema in which the module is defined.

Note: Prior to Oracle 8*i*, all PL/SQL stored procedures and packages were definer rights modules.

Invoker Rights Modules

Invoker rights modules run under the privileges of the invoker of the module. Therefore, DBMS_SQL subprograms called from invoker rights modules run under the privileges of the invoker of the module.

When a module has AUTHID set to current_user, the unqualified names are resolved with respect to the invoker's schema.

Example:

income is an invoker rights stored procedure in USER1's schema, and USER2 has been granted EXECUTE privilege on it.

```
CREATE PROCEDURE income(amount number)
   AUTHID current_user IS
   c number;
   n number;
BEGIN
   c:= dbms_sql.open_cursor;
   dbms_sql.parse(c, 'insert into accts(''income'', :1)', dbms_sql.native);
   dbms_sql.bind_variable(c, '1', amount);
   n := dbms_sql.execute(c);
   dbms_sql.close_cursor(c);
END;
```

If USER1 calls USER1.income, then USER1's privileges are used, and name resolution of unqualified names is done with respect to USER1's schema.

If USER2 calls USER1.income, then USER2's privileges are used, and name resolution of unqualified names (such as accts) is done with respect to USER2's schema.

See Also: *PL/SQL User's Guide and Reference*

Anonymous Blocks

Any DBMS_SQL subprograms called from an anonymous PL/SQL block are run using the privileges of the current user.

Processing Queries

If you are using dynamic SQL to process a query, then you must perform the following steps:

1. Specify the variables that are to receive the values returned by the SELECT statement by calling DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY.
2. Run your SELECT statement by calling EXECUTE.
3. Call FETCH_ROWS (or EXECUTE_AND_FETCH) to retrieve the rows that satisfied your query.

4. Call COLUMN_VALUE or COLUMN_VALUE_LONG to determine the value of a column retrieved by the FETCH_ROWS call for your query. If you used anonymous blocks containing calls to PL/SQL procedures, then you must call VARIABLE_VALUE to retrieve the values assigned to the output variables of these procedures.

Examples

This section provides example procedures that make use of the DBMS_SQL package.

Example 1

The following sample procedure is passed a SQL statement, which it then parses and runs:

```
CREATE OR REPLACE PROCEDURE exec(STRING IN varchar2) AS
    cursor_name INTEGER;
    ret INTEGER;
BEGIN
   cursor_name := DBMS_SQL.OPEN_CURSOR;
```

DDL statements are run by the parse call, which performs the implied commit.

```
   DBMS_SQL.PARSE(cursor_name, string, DBMS_SQL.native);
   ret := DBMS_SQL.EXECUTE(cursor_name);
   DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Creating such a procedure enables you to perform the following operations:

- The SQL statement can be dynamically generated at runtime by the calling program.
- The SQL statement can be a DDL statement or a DML without binds.

For example, after creating this procedure, you could make the following call:

```
exec('create table acct(c1 integer)');
```

You could even call this procedure remotely, as shown in the following example. This lets you perform remote DDL.

```
exec@hq.com('CREATE TABLE acct(c1 INTEGER)');
```

Example 2

The following sample procedure is passed the names of a source and a destination table, and copies the rows from the source table to the destination table. This sample procedure assumes that both the source and destination tables have the following columns:

```
id        of type NUMBER
name      of type VARCHAR2(30)
birthdate of type DATE
```

This procedure does not specifically require the use of dynamic SQL; however, it illustrates the concepts of this package.

```
CREATE OR REPLACE PROCEDURE copy (
     source      IN VARCHAR2,
     destination IN VARCHAR2) IS
     id_var             NUMBER;
     name_var           VARCHAR2(30);
     birthdate_var      DATE;
     source_cursor      INTEGER;
     destination_cursor INTEGER;
     ignore             INTEGER;
  BEGIN

  -- Prepare a cursor to select from the source table:
     source_cursor := dbms_sql.open_cursor;
     DBMS_SQL.PARSE(source_cursor,
         'SELECT id, name, birthdate FROM ' || source,
          DBMS_SQL.native);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 1, id_var);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 2, name_var, 30);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 3, birthdate_var);
     ignore := DBMS_SQL.EXECUTE(source_cursor);

  -- Prepare a cursor to insert into the destination table:
     destination_cursor := DBMS_SQL.OPEN_CURSOR;
     DBMS_SQL.PARSE(destination_cursor,
                 'INSERT INTO ' || destination ||
                 ' VALUES (:id_bind, :name_bind, :birthdate_bind)',
                  DBMS_SQL.native);

  -- Fetch a row from the source table and insert it into the destination table:
     LOOP
       IF DBMS_SQL.FETCH_ROWS(source_cursor)>0 THEN
         -- get column values of the row
```

```
        DBMS_SQL.COLUMN_VALUE(source_cursor, 1, id_var);
        DBMS_SQL.COLUMN_VALUE(source_cursor, 2, name_var);
        DBMS_SQL.COLUMN_VALUE(source_cursor, 3, birthdate_var);

  -- Bind the row into the cursor that inserts into the destination table. You
  -- could alter this example to require the use of dynamic SQL by inserting an
  -- if condition before the bind.
      DBMS_SQL.BIND_VARIABLE(destination_cursor, ':id_bind', id_var);
      DBMS_SQL.BIND_VARIABLE(destination_cursor, ':name_bind', name_var);
      DBMS_SQL.BIND_VARIABLE(destination_cursor, ':birthdate_bind',
birthdate_var);
      ignore := DBMS_SQL.EXECUTE(destination_cursor);
    ELSE

  -- No more rows to copy:
      EXIT;
    END IF;
  END LOOP;

  -- Commit and close all cursors:
    COMMIT;
    DBMS_SQL.CLOSE_CURSOR(source_cursor);
    DBMS_SQL.CLOSE_CURSOR(destination_cursor);
 EXCEPTION
    WHEN OTHERS THEN
      IF DBMS_SQL.IS_OPEN(source_cursor) THEN
        DBMS_SQL.CLOSE_CURSOR(source_cursor);
      END IF;
      IF DBMS_SQL.IS_OPEN(destination_cursor) THEN
        DBMS_SQL.CLOSE_CURSOR(destination_cursor);
      END IF;
      RAISE;
  END;
/
```

Examples 3, 4, and 5: Bulk DML

This series of examples shows how to use bulk array binds (table items) in the SQL DML statements DELETE, INSERT, and UPDATE.

In a DELETE statement, for example, you could bind in an array in the WHERE clause and have the statement be run for each element in the array:

```
declare
  stmt varchar2(200);
  dept_no_array dbms_sql.Number_Table;
```

```
  c number;
  dummy number;
begin
  dept_no_array(1) := 10; dept_no_array(2) := 20;
  dept_no_array(3) := 30; dept_no_array(4) := 40;
  dept_no_array(5) := 30; dept_no_array(6) := 40;
  stmt := 'delete from emp where deptno = :dept_array';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':dept_array', dept_no_array, 1, 4);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

In the preceding example, only elements 1 through 4 are used as specified by the bind_array call. Each element of the array potentially deletes a large number of employees from the database.

Here is an example of a bulk `INSERT` statement:

```
declare
  stmt varchar2(200);
  empno_array dbms_sql.Number_Table;
  empname_array dbms_sql.Varchar2_Table;
  c number;
  dummy number;
begin
  for i in 0..9 loop
    empno_array(i) := 1000 + i;
    empname_array(I) := get_name(i);
  end loop;
  stmt := 'insert into emp values(:num_array, :name_array)';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':num_array', empno_array);
  dbms_sql.bind_array(c, ':name_array', empname_array);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);
```

```
  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

When the execute takes place, all 10 of the employees are inserted into the table.

Finally, here is an example of an bulk UPDATE statement.

```
declare
  stmt varchar2(200);
  emp_no_array dbms_sql.Number_Table;
  emp_addr_array dbms_sql.Varchar2_Table;
  c number;
  dummy number;
begin
  for i in 0..9 loop
    emp_no_array(i) := 1000 + i;
    emp_addr_array(I) := get_new_addr(i);
  end loop;
  stmt := 'update emp set ename = :name_array
    where empno = :num_array';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':num_array', empno_array);
  dbms_sql.bind_array(c, ':name_array', empname_array);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

When the EXECUTE call happens, the addresses of all employees are updated at once. The two collections are always stepped in unison. If the WHERE clause returns more than one row, then all those employees get the address the addr_array happens to be pointing to at that time.

Examples 6 and 7: Defining an Array

The following examples show how to use the DEFINE_ARRAY procedure:

```
declare
  c       number;
  d       number;
  n_tab   dbms_sql.Number_Table;
  indx    number := -10;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'select n from t order by 1', dbms_sql);

  dbms_sql.define_array(c, 1, n_tab, 10, indx);

  d := dbms_sql.execute(c);
  loop
    d := dbms_sql.fetch_rows(c);

    dbms_sql.column_value(c, 1, n_tab);

    exit when d != 10;
  end loop;

  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

Each time the preceding example does a FETCH_ROWS call, it fetches 10 rows that are kept in DBMS_SQL buffers. When the COLUMN_VALUE call is run, those rows move into the PL/SQL table specified (in this case n_tab), at positions -10 to -1, as specified in the DEFINE statements. When the second batch is fetched in the loop, the rows go to positions 0 to 9; and so on.

A current index into each array is maintained automatically. This index is initialized to "indx" at EXECUTE and keeps getting updated every time a COLUMN_VALUE call is made. If you reexecute at any point, then the current index for each DEFINE is re-initialized to "indx".

In this way the entire result of the query is fetched into the table. When FETCH_ROWS cannot fetch 10 rows, it returns the number of rows actually fetched (if no rows could be fetched, then it returns zero) and exits the loop.

Here is another example of using the DEFINE_ARRAY procedure:

Consider a table MULTI_TAB defined as:

```
create table multi_tab (num number,
                        dat1 date,
                        var varchar2(24),
                        dat2 date)
```

To select everything from this table and move it into four PL/SQL tables, you could use the following simple program:

```
declare
  c       number;
  d       number;
  n_tab  dbms_sql.Number_Table;
  d_tab1 dbms_sql.Date_Table;
  v_tab  dbms_sql.Varchar2_Table;
  d_tab2 dbms_sql.Date_Table;
  indx number := 10;
begin

  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'select * from multi_tab order by 1', dbms_sql);

  dbms_sql.define_array(c, 1, n_tab,  5, indx);
  dbms_sql.define_array(c, 2, d_tab1, 5, indx);
  dbms_sql.define_array(c, 3, v_tab,  5, indx);
  dbms_sql.define_array(c, 4, d_tab2, 5, indx);

  d := dbms_sql.execute(c);

  loop
    d := dbms_sql.fetch_rows(c);

    dbms_sql.column_value(c, 1, n_tab);
    dbms_sql.column_value(c, 2, d_tab1);
    dbms_sql.column_value(c, 3, v_tab);
    dbms_sql.column_value(c, 4, d_tab2);

    exit when d != 5;
  end loop;
```

```
  dbms_sql.close_cursor(c);

/*
```

The four tables can be used for anything. One usage might be to use BIND_ARRAY to move the rows to another table by using a query such as 'INSERT into SOME_T values (:a, :b, :c, :d);

```
*/

exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

Example 8: Describe Columns

This can be used as a substitute to the SQL*Plus DESCRIBE call by using a SELECT * query on the table that you want to describe.

```
declare
  c number;
  d number;
  col_cnt integer;
  f boolean;
  rec_tab dbms_sql.desc_tab;
  col_num number;
  procedure print_rec(rec in dbms_sql.desc_rec) is
  begin
    dbms_output.new_line;
    dbms_output.put_line('col_type            =    '
                         || rec.col_type);
    dbms_output.put_line('col_maxlen          =    '
                         || rec.col_max_len);
    dbms_output.put_line('col_name            =    '
                         || rec.col_name);
    dbms_output.put_line('col_name_len        =    '
                         || rec.col_name_len);
    dbms_output.put_line('col_schema_name     =    '
                         || rec.col_schema_name);
    dbms_output.put_line('col_schema_name_len =    '
                         || rec.col_schema_name_len);
```

```
    dbms_output.put_line('col_precision       =    '
                         || rec.col_precision);
    dbms_output.put_line('col_scale           =    '
                         || rec.col_scale);
    dbms_output.put('col_null_ok         =    ');
    if (rec.col_null_ok) then
      dbms_output.put_line('true');
    else
      dbms_output.put_line('false');
    end if;
  end;
begin
  c := dbms_sql.open_cursor;

  dbms_sql.parse(c, 'select * from scott.bonus', dbms_sql);

  d := dbms_sql.execute(c);

  dbms_sql.describe_columns(c, col_cnt, rec_tab);

/*
 * Following loop could simply be for j in 1..col_cnt loop.
 * Here we are simply illustrating some of the PL/SQL table
 * features.
 */
  col_num := rec_tab.first;
  if (col_num is not null) then
    loop
      print_rec(rec_tab(col_num));
      col_num := rec_tab.next(col_num);
      exit when (col_num is null);
    end loop;
  end if;

  dbms_sql.close_cursor(c);
end;
/
```

Example 9: RETURNING clause The `RETURNING` clause was added to DML statements in Oracle 8.0.3. With this clause, `INSERT`, `UPDATE`, and `DELETE` statements can return values of expressions. These values are returned in bind variables.

`DBMS_SQL.BIND_VARIABLE` is used to bind these outbinds if a single row is inserted, updated, or deleted. If multiple rows are inserted, updated, or deleted,

then DBMS_SQL.BIND_ARRAY is used. DBMS_SQL.VARIABLE_VALUE must be called to get the values in these bind variables.

> **Note:** This is similar to DBMS_SQL.VARIABLE_VALUE, which must be called after running a PL/SQL block with an out-bind inside DBMS_SQL.

i) Single row insert

```
create or replace procedure single_Row_insert
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'insert into tab values (:bnd1, :bnd2) ' ||
                    'returning c1*c2 into :bnd3', 2);
dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r); -- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

ii) Single row update

```
create or replace procedure single_Row_update
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'update tab set c1 = :bnd1, c2 = :bnd2 ' ||
                    'where rownum < 2' ||
                    'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
```

```
/
```

iii) Single row delete

```
create or replace procedure single_Row_Delete
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'delete from tab ' ||
                    'where rownum < 2 ' ||
                   'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

iv) Multi-row insert

```
create or replace procedure multi_Row_insert
     (c1 dbms_sql.number_table, c2 dbms_sql.number_table,
      r out dbms_sql.number_table) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'insert into tab values (:bnd1, :bnd2) ' ||
                    'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_array(c, 'bnd1', c1);
  dbms_sql.bind_array(c, 'bnd2', c2);
  dbms_sql.bind_array(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

v) Multi row Update.

```
create or replace procedure multi_Row_update
     (c1 number, c2 number, r out dbms_Sql.number_table) is
```

```
 c number;
 n number;
begin
   c := dbms_sql.open_cursor;
   dbms_sql.parse(c, 'update tab set c1 = :bnd1 where c2 = :bnd2 ' ||
                     'returning c1*c2 into :bnd3', 2);
   dbms_sql.bind_variable(c, 'bnd1', c1);
   dbms_sql.bind_variable(c, 'bnd2', c2);
   dbms_sql.bind_array(c, 'bnd3', r);
   n := dbms_sql.execute(c);
   dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
   dbms_Sql.close_Cursor(c);
 end;
 /
```

Note: bnd1 and bnd2 can be array as well. The value of the expression for all the rows updated will be in bnd3. There is no way of differentiating which rows got updated of each value of bnd1 and bnd2.

vi) Multi-row delete

```
create or replace procedure multi_row_delete
     (c1 dbms_Sql.number_table,
      r out dbms_sql.number_table) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'delete from tab where c1 = :bnd1' ||
                    'returning c1*c2 into :bnd2', 2);
  dbms_sql.bind_array(c, 'bnd1', c1);
  dbms_sql.bind_array(c, 'bnd2', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd2', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

vii) Out-bind in bulk PL/SQL

```
create or replace foo (n number, square out number) is
begin square := n * n; end;/
```

```
create or replace procedure bulk_plsql
   (n dbms_sql.number_Table, square out dbms_sql.number_table) is
c number;
r number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'begin foo(:bnd1, :bnd2); end;', 2);
  dbms_sql.bind_array(c, 'bnd1', n);
  dbms_Sql.bind_Array(c, 'bnd2', square);
  r := dbms_sql.execute(c);
  dbms_Sql.variable_Value(c, 'bnd2', square);
end;
/
```

Note: DBMS_SQL.BIND_ARRAY of number_Table internally binds a number. The number of times statement is run depends on the number of elements in an inbind array.

Processing Updates, Inserts, and Deletes

If you are using dynamic SQL to process an INSERT, UPDATE, or DELETE, then you must perform the following steps:

1. You must first run your INSERT, UPDATE, or DELETE statement by calling EXECUTE.
2. If statements have the returning clause, then you must call VARIABLE_VALUE to retrieve the values assigned to the output variables.

Locating Errors

There are additional functions in the DBMS_SQL package for obtaining information about the last referenced cursor in the session. The values returned by these functions are only meaningful immediately after a SQL statement is run. In addition, some error-locating functions are only meaningful after certain DBMS_SQL calls. For example, you call LAST_ERROR_POSITION immediately after a PARSE.

Summary of DBMS_SQL Subprograms

Table 69–1 DBMS_SQL Subprograms

Subprogram	Description
OPEN_CURSOR Function on page 69-24	Returns cursor ID number of new cursor.
PARSE Procedure on page 69-24	Parses given statement.
BIND_VARIABLE and BIND_ ARRAY Procedures on page 69-27	Binds a given value to a given variable.
BIND_VARIABLE and BIND_ ARRAY Procedures on page 69-27	Binds a given value to a given collection.
DEFINE_COLUMN Procedure on page 69-31	Defines a column to be selected from the given cursor, used only with `SELECT` statements.
DEFINE_ARRAY Procedure on page 69-33	Defines a collection to be selected from the given cursor, used only with `SELECT` statements.
DEFINE_COLUMN_LONG Procedure on page 69-35	Defines a `LONG` column to be selected from the given cursor, used only with `SELECT` statements.
EXECUTE Function on page 69-36	Executes a given cursor.
EXECUTE_AND_FETCH Function on page 69-36	Executes a given cursor and fetch rows.
FETCH_ROWS Function on page 69-37	Fetches a row from a given cursor.
COLUMN_VALUE Procedure on page 69-38	Returns value of the cursor element for a given position in a cursor.
COLUMN_VALUE_LONG Procedure on page 69-40	Returns a selected part of a `LONG` column, that has been defined using `DEFINE_COLUMN_LONG`.
VARIABLE_VALUE Procedure on page 69-41	Returns value of named variable for given cursor.
IS_OPEN Function on page 69-43	Returns `TRUE` if given cursor is open.
DESCRIBE_COLUMNS Procedure on page 69-44	Describes the columns for a cursor opened and parsed through `DBMS_SQL`.
CLOSE_CURSOR Procedure on page 69-46	Closes given cursor and frees memory.
LAST_ERROR_POSITION Function on page 69-47	Returns byte offset in the SQL statement text where the error occurred.

Table 69–1 DBMS_SQL Subprograms

Subprogram	Description
LAST_ROW_COUNT Function on page 69-47	Returns cumulative count of the number of rows fetched.
LAST_ROW_ID Function on page 69-47	Returns ROWID of last row processed.
LAST_SQL_FUNCTION_CODE Function on page 69-48	Returns SQL function code for statement.

OPEN_CURSOR Function

This procedure opens a new cursor. When you no longer need this cursor, you must close it explicitly by calling CLOSE_CURSOR.

You can use cursors to run the same SQL statement repeatedly or to run a new SQL statement. When a cursor is reused, the contents of the corresponding cursor data area are reset when the new SQL statement is parsed. It is never necessary to close and reopen a cursor before reusing it.

Syntax

```
DBMS_SQL.OPEN_CURSOR
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(open_cursor,RNDS,WNDS);
```

Returns

This function returns the cursor ID number of the new cursor.

PARSE Procedure

This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.

There are two versions of the PARSE procedure: one uses a VARCHAR2 statement as an argument, and the other uses a VARCHAR2S (table of VARCHAR2) as an argument.

Caution: Using DBMS_SQL to dynamically run DDL statements can result in the program hanging. For example, a call to a procedure in a package results in the package being locked until the execution returns to the user side. Any operation that results in a conflicting lock, such as dynamically trying to drop the package before the first lock is released, results in a hang.

The size limit for parsing SQL statements with the preceding syntax is 32KB.

Syntax

```
DBMS_SQL.PARSE (
   c                IN INTEGER,
   statement        IN VARCHAR2,
   language_flag    IN INTEGER);
```

The PARSE procedure also supports the following syntax for large SQL statements:

Note: The procedure concatenates elements of a PL/SQL table statement and parses the resulting string. You can use this procedure to parse a statement that is longer than the limit for a single VARCHAR2 variable by splitting up the statement.

```
DBMS_SQL.PARSE (
   c                IN INTEGER,
   statement        IN VARCHAR2S,
   lb               IN INTEGER,
   ub               IN INTEGER,
   lfflg            IN BOOLEAN,
   language_flag    IN INTEGER);
```

Parameters

Table 69–2 PARSE Procedure Parameters

Parameter	Description
c	ID number of the cursor in which to parse the statement.

statement	SQL statement to be parsed.
	Unlike PL/SQL statements, your SQL statement should not include a final semicolon. For example:
	`DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2);`
	`DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab values(1)', 2);`

lb	Lower bound for elements in the statement.
ub	Upper bound for elements in the statement.
lfflg	If TRUE, then insert a linefeed after each element on concatenation.
language_flag	Determines how Oracle handles the SQL statement. The following options are recognized: ■ V6 (or 0) specifies version 6 behavior. ■ NATIVE (or 1) specifies normal behavior for the database to which the program is connected. ■ V7 (or 2) specifies Oracle7 behavior.

Note: Because client-side code cannot reference remote package variables or constants, you must explicitly use the values of the constants.

For example, the following code does *not* compile on the client:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, dbms_sql.V7); -- uses
constant dbms_sql.V7
```

The following code works on the client, because the argument is explicitly provided:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, 2); -- compiles on
the client
```

Example 9: VARCHAR2S Datatype for Parsing Large SQL Strings

To parse SQL statements larger than 32 KB, DBMS_SQL makes use of PL/SQL tables to pass a table of strings to the PARSE procedure. These strings are concatenated and then passed on to the Oracle server.

You can declare a local variable as the VARCHAR2S table-item type, and then use the PARSE procedure to parse a large SQL statement as VARCHAR2S.

The definition of the VARCHAR2S datatype is:

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
```

Exceptions

If you create a type/procedure/function/package using DBMS_SQL that has compilation warnings, an ORA-24344 exception is raised, and the procedure is still created.

BIND_VARIABLE and BIND_ARRAY Procedures

These two procedures bind a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement. If the variable is an IN or IN/OUT variable or an IN collection, then the given bind value must be valid for the variable or array type. Bind values for OUT variables are ignored.

The bind variables or collections of a SQL statement are identified by their names. When binding a value to a bind variable or bind array, the string identifying it in the statement must contain a leading colon, as shown in the following example:

```
SELECT emp_name FROM emp WHERE SAL > :X;
```

For this example, the corresponding bind call would look similar to

```
BIND_VARIABLE(cursor_name, ':X', 3500);
```

or

```
BIND_VARIABLE (cursor_name, 'X', 3500);
```

Syntax

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
```

```
   value          IN <datatype>)
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
UROWID
```

Notice that BIND_VARIABLE is overloaded to accept different datatypes.

> **See Also:** *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(bind_variable,WNDS);
```

Usage Notes

The following syntax is also supported for BIND_VARIABLE. The square brackets [] indicate an optional parameter for the BIND_VARIABLE function.

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN
INTEGER]);
```

To bind CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.BIND_VARIABLE_CHAR (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN CHAR CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_RAW (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN RAW [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_ROWID (
```

```
c               IN INTEGER,
name            IN VARCHAR2,
value           IN ROWID);
```

Parameters

Table 69–3 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the variable in the statement.
value	Value that you want to bind to the variable in the cursor. For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.
out_value_size	Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable. If no size is given, then the length of the current value is used. This parameter must be specified if the value parameter is not initialized.

Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The DBMS_SQL package lets you work on collections of data using the PL/SQL table type.

Table items are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local DBMS_SQL buffers (the same as for all scalar types) and then the table is manipulated from the

local DBMS_SQL buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.

Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS_SQL.

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
type Urowid_Table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
```

Syntax

```
DBMS_SQL.BIND_ARRAY (
   c                  IN INTEGER,
   name               IN VARCHAR2,
   <table_variable>   IN <datatype>
 [,index1             IN INTEGER,
   index2             IN INTEGER)] );
```

Where the <table_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
<urowid_tab>   Urowid_Table
```

Notice that the BIND_ARRAY procedure is overloaded to accept different datatypes.

Parameters

Table 69–4 BIND_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the collection in the statement.
table_variable	Local variable that has been declared as <datatype>.
index1	Index for the table element that marks the lower bound of the range.
index2	Index for the table element that marks the upper bound of the range.

Usage Notes

For binding a range, the table must contain the elements that specify the range — tab(index1) and tab(index2) — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between tab(index1) and tab(index2) are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

> **See Also:** "Examples 3, 4, and 5: Bulk DML" on page 69-12 for examples of how to bind collections.

DEFINE_COLUMN Procedure

This procedure defines a column to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN <datatype>)
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
UROWID
```

Notice that DEFINE_COLUMN is overloaded to accept different datatypes.

> **See Also:** *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

The following syntax is also supported for the DEFINE_COLUMN procedure:

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN VARCHAR2 CHARACTER SET ANY_CS,
   column_size    IN INTEGER),
   urowid         IN INTEGER;
```

To define columns with CHAR, RAW, and ROWID data, you can use the following variations on the procedure syntax:

```
DBMS_SQL.DEFINE_COLUMN_CHAR (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN CHAR CHARACTER SET ANY_CS,
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_RAW (
   c              IN INTEGER,
   position       IN INTEGER,
```

```
   column         IN RAW,
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_ROWID (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN ROWID);
```

Parameters

Table 69–5 DEFINE_COLUMN Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.
column	Value of the column being defined. The type of this value determines the type for the column being defined.
column_size	Maximum expected size of the column value, in bytes, for columns of type VARCHAR2, CHAR, and RAW.

DEFINE_ARRAY Procedure

This procedure defines the collection for column into which you want to fetch rows (with a FETCH_ROWS call). This procedure lets you do batch fetching of rows from a single SELECT statement. A single fetch call brings over a number of rows into the PL/SQL aggregate object.

When you fetch the rows, they are copied into DBMS_SQL buffers until you run a COLUMN_VALUE call, at which time the rows are copied into the table that was passed as an argument to the COLUMN_VALUE call.

Scalar and LOB Types for Collections

You can declare a local variable as one of the following table-item types, and then fetch any number of rows into it using DBMS_SQL. (These are the same types as you can specify for the BIND_ARRAY procedure.)

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
```

```
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
type Urowid_Table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;
```

Syntax

```
DBMS_SQL.DEFINE_ARRAY (
   c           IN INTEGER,
   position    IN INTEGER,
   <table_variable>    IN <datatype>
   cnt         IN INTEGER,
   lower_bnd   IN INTEGER);
```

Where <table_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
<urowid_tab>   Urowid_Table
```

Notice that DEFINE_ARRAY is overloaded to accept different datatypes.

Pragmas

```
pragma restrict_references(define_array,RNDS,WNDS);
```

The subsequent FETCH_ROWS call fetch "count" rows. When the COLUMN_VALUE call is made, these rows are placed in positions indx, indx+1, indx+2, and so on. While there are still rows coming, the user keeps issuing FETCH_ROWS / COLUMN_VALUE calls. The rows keep accumulating in the table specified as an argument in the COLUMN_VALUE call.

Parameters

Table 69–6 DEFINE_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind an array.
position	Relative position of the column in the array being defined. The first column in a statement has position 1.
table_variable	Local variable that has been declared as <datatype>.
cnt	Number of rows that must be fetched.
lower_bnd	Results are copied into the collection, starting at this lower bound index.

The count (cnt) must be an integer greater than zero; otherwise an exception is raised. The indx can be positive, negative, or zero. A query on which a DEFINE_ARRAY call was issued cannot contain array binds.

> **See Also:** "Examples 6 and 7: Defining an Array" on page 69-15 for examples of how to define collections.

DEFINE_COLUMN_LONG Procedure

This procedure defines a LONG column for a SELECT cursor. The column being defined is identified by its relative position in the SELECT list of the statement for the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN_LONG (
   c             IN INTEGER,
   position      IN INTEGER);
```

Parameters

Table 69–7 DEFINE_COLUMN_LONG Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.

EXECUTE Function

This function executes a given cursor. This function accepts the ID number of the cursor and returns the number of rows processed. The return value is only valid for INSERT, UPDATE, and DELETE statements; for other types of statements, including DDL, the return value is undefined and should be ignored.

Syntax

```
DBMS_SQL.EXECUTE (
   c   IN INTEGER)
  RETURN INTEGER;
```

Parameters

Table 69–8 EXECUTE Function Parameters

Parameter	Description
c	Cursor ID number of the cursor to execute.

EXECUTE_AND_FETCH Function

This function executes the given cursor and fetches rows. This function provides the same functionality as calling EXECUTE and then calling FETCH_ROWS. Calling EXECUTE_AND_FETCH instead, however, may reduce the number of network round-trips when used against a remote database.

The EXECUTE_AND_FETCH function returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.EXECUTE_AND_FETCH (
   c              IN INTEGER,
   exact          IN BOOLEAN DEFAULT FALSE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(execute_and_fetch,WNDS);
```

Parameters

Table 69–9 EXECUTE_AND_FETCH Function Parameters

Parameter	Description
c	ID number of the cursor to execute and fetch.
exact	Set to TRUE to raise an exception if the number of rows actually matching the query differs from one. Note: Oracle does not support the exact fetch TRUE option with LONG columns. Even if an exception is raised, the rows are still fetched and available.

FETCH_ROWS Function

This function fetches a row from a given cursor. You can call FETCH_ROWS repeatedly as long as there are rows remaining to be fetched. These rows are retrieved into a buffer, and must be read by calling COLUMN_VALUE, for each column, after each call to FETCH_ROWS.

The FETCH_ROWS function accepts the ID number of the cursor to fetch, and returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.FETCH_ROWS (
   c              IN INTEGER)
  RETURN INTEGER;
```

Parameters

Table 69–10 FETCH_ROWS Function Parameters

Parameter	Description
c	ID number.

Pragmas

```
pragma restrict_references(fetch_rows,WNDS);
```

COLUMN_VALUE Procedure

This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling FETCH_ROWS.

Syntax

```
DBMS_SQL.COLUMN_VALUE (
   c                IN  INTEGER,
   position         IN  INTEGER,
   value            OUT <datatype>
 [,column_error     OUT NUMBER]
 [,actual_length    OUT INTEGER]);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
UROWID
```

Note: The square brackets [ ] indicate optional parameters.

See Also: *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(column_value,RNDS,WNDS);
```

The following syntax is also supported for the COLUMN_VALUE procedure:

```
DBMS_SQL.COLUMN_VALUE(
   c                IN  INTEGER,
   position         IN  INTEGER,
   <table_variable> IN  <datatype>);
```

Where the <table_variable> and its corresponding <datatype> can be any one of these matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
<urowid_tab>   Urowid_Table
```

For columns containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.COLUMN_VALUE_CHAR (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT CHAR CHARACTER SET ANY_CS
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_RAW (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT RAW
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_ROWID (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT ROWID
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);
```

Parameters

Table 69–11 COLUMN_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which you are fetching the values.
position	Relative position of the column in the cursor. The first column in a statement has position 1.
value	Returns the value at the specified column and row. If the row number specified is greater than the total number of rows fetched, then you receive an error message. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to DEFINE_COLUMN.
table_ variable	Local variable that has been declared <datatype>.
column_error	Returns any error code for the specified column value.
actual_length	The actual length, before any truncation, of the value in the specified column.

Exceptions:

inconsistent_type (ORA-06562) is raised if the type of the given OUT parameter value is different from the actual type of the value. This type was the given type when the column was defined by calling procedure DEFINE_COLUMN.

COLUMN_VALUE_LONG Procedure

This procedure gets part of the value of a long column.

Syntax

```
DBMS_SQL.COLUMN_VALUE_LONG (
   c            IN  INTEGER,
   position     IN  INTEGER,
   length       IN  INTEGER,
   offset       IN  INTEGER,
   value        OUT VARCHAR2,
   value_length OUT INTEGER);
```

Pragmas

```
pragma restrict_references(column_value_long,RNDS,WNDS);
```

Parameters

Table 69–12 COLUMN_VALUE_LONG Procedure Parameters

Parameter	Description
c	Cursor ID number of the cursor from which to get the value.
position	Position of the column of which to get the value.
length	Number of bytes of the long value to fetch.
offset	Offset into the long field for start of fetch.
value	Value of the column as a VARCHAR2.
value_length	Number of bytes actually returned in value.

VARIABLE_VALUE Procedure

This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with `returning` clause.

Syntax

```
DBMS_SQL.VARIABLE_VALUE (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT <datatype>);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
UROWID
```

Pragmas

```
pragma restrict_references(variable_value,RNDS,WNDS);
```

The following syntax is also supported for the VARIABLE_VALUE procedure:

```
DBMS_SQL.VARIABLE_VALUE (
   c                IN  INTEGER,
   name             IN  VARCHAR2,
   <table_variable>  IN  <datatype>);
```

Where the <table_variable> and its corresponding <datatype> can be any one of these matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
<urowid_tab>   Urowid_Table
```

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.VARIABLE_VALUE_CHAR (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT CHAR CHARACTER SET ANY_CS);

DBMS_SQL.VARIABLE_VALUE_RAW (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT RAW);

DBMS_SQL.VARIABLE_VALUE_ROWID (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT ROWID);
```

Parameters

Table 69–13 VARIABLE_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which to get the values.
name	Name of the variable for which you are retrieving the value.
value	Returns the value of the variable for the specified position. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE.
position	Relative position of the column in the cursor. The first column in a statement has position 1.

IS_OPEN Function

This function checks to see if the given cursor is currently open.

Syntax

```
DBMS_SQL.IS_OPEN (
   c              IN INTEGER)
  RETURN BOOLEAN;
```

Pragmas

```
pragma restrict_references(is_open,RNDS,WNDS);
```

Parameters

Table 69–14 IS_OPEN Function Parameters

Parameter	Description
c	Cursor ID number of the cursor to check.

Returns

Table 69–15 IS_OPEN Function Return Values

Return Value	Description
TRUE	Given cursor is currently open.
FALSE	Given cursor is currently not open.

DESCRIBE_COLUMNS Procedure

This procedure describes the columns for a cursor opened and parsed through DBMS_SQL.

The DESC_REC Type

The DBMS_SQL package declares the DESC_REC record type as follows:

```
type desc_rec is record (
   col_type            BINARY_INTEGER := 0,
   col_max_len         BINARY_INTEGER := 0,
   col_name            VARCHAR2(32)   := '',
   col_name_len        BINARY_INTEGER := 0,
   col_schema_name     VARCHAR2(32)   := '',
   col_schema_name_len BINARY_INTEGER := 0,
   col_precision       BINARY_INTEGER := 0,
   col_scale           BINARY_INTEGER := 0,
   col_charsetid       BINARY_INTEGER := 0,
   col_charsetform     BINARY_INTEGER := 0,
   col_null_ok         BOOLEAN        := TRUE);
```

Parameters

Table 69–16 DESC_REC Type Parameters

Parameter	Description
col_type	Type of the column being described.
col_max_len	Maximum length of the column.
col_name	Name of the column.
col_name_len	Length of the column name.
col_schema_name	Name of the schema the column type was defined in, if an object type.
col_schema_name_len	Length of the schema.
col_precision	Column precision, if a number.
col_scale	Column scale, if a number.
col_charsetid	Column character set identifier.
col_charsetform	Column character set form.
col_null_ok	True if column can be null.

The DESC_TAB Type

The DESC_TAB type is a PL/SQL table of DESC_REC records:

```
type desc_tab is table of desc_rec index by BINARY_INTEGER;
```

You can declare a local variable as the PL/SQL table type DESC_TAB, and then call the DESCRIBE_COLUMNS procedure to fill in the table with the description of each column. All columns are described; you cannot describe a single column.

Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS (
   c             IN  INTEGER,
   col_cnt       OUT INTEGER,
   desc_t        OUT DESC_TAB);
```

Parameters

Table 69–17 DBMS_SQL.DESCRIBE_COLUMNS Procedure Parameters

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_t	Table of DESC_REC, each DESC_REC describing a column in the query.

See Also: "Example 8: Describe Columns" on page 69-17 illustrates how to use DESCRIBE_COLUMNS.

CLOSE_CURSOR Procedure

This procedure closes a given cursor.

Syntax

```
DBMS_SQL.CLOSE_CURSOR (
   c    IN OUT INTEGER);
```

Pragmas

```
pragma restrict_references(close_cursor,RNDS,WNDS);
```

Parameters

Table 69–18 CLOSE_CURSOR Procedure Parameters

Parameter	Mode	Description
c	IN	ID number of the cursor that you want to close.
c	OUT	Cursor is set to null. After you call CLOSE_CURSOR, the memory allocated to the cursor is released and you can no longer fetch from that cursor.

LAST_ERROR_POSITION Function

This function returns the byte offset in the SQL statement text where the error occurred. The first character in the SQL statement is at position 0.

Syntax

```
DBMS_SQL.LAST_ERROR_POSITION
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_error_position,RNDS,WNDS);
```

Usage Notes

Call this function after a PARSE call, before any other DBMS_SQL procedures or functions are called.

LAST_ROW_COUNT Function

This function returns the cumulative count of the number of rows fetched.

Syntax

```
DBMS_SQL.LAST_ROW_COUNT
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_row_count,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call. If called after an EXECUTE call, then the value returned is zero.

LAST_ROW_ID Function

This function returns the ROWID of the last row processed.

Syntax

```
DBMS_SQL.LAST_ROW_ID
   RETURN ROWID;
```

Pragmas

```
pragma restrict_references(last_row_id,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call.

LAST_SQL_FUNCTION_CODE Function

This function returns the SQL function code for the statement. These codes are listed in the *Oracle Call Interface Programmer's Guide*.

Syntax

```
DBMS_SQL.LAST_SQL_FUNCTION_CODE
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_sql_function_code,RNDS,WNDS);
```

Usage Notes

You should call this function immediately after the SQL statement is run; otherwise, the return value is undefined.

70

DBMS_STATS

With DBMS_STATS you can view and modify optimizer statistics gathered for database objects. The statistics can reside in the dictionary or in a table created in the user's schema for this purpose. You can also collect and manage user-defined statistics for tables and domain indexes using this package. For example, if the DELETE_COLUMN_STATS procedure is invoked on a column for which an association is defined, user-defined statistics for that column are deleted in addition to deletion of the standard statistics.

Only statistics stored in the dictionary have an impact on the cost-based optimizer. You can also use DBMS_STATS to gather statistics in parallel.

This chapter contains the following topics:

Using DBMS_STATS

The DBMS_STATS subprograms perform the following general functions:

- Set or get statistics
- Transfer statistics
- Gather optimizer statistics

Most of the DBMS_STATS procedures include the three parameters statown, stattab, and statid. These parameters allow you to store statistics in your own tables (outside of the dictionary), which does not affect the optimizer. Therefore, you can maintain and experiment with *sets* of statistics.

The stattab parameter specifies the name of a table in which to hold statistics, and it is assumed that it resides in the same schema as the object for which statistics are collected (unless the statown parameter is specified). You can create multiple tables with different stattab identifiers to hold separate sets of statistics.

Additionally, you can maintain different sets of statistics within a single stattab by using the statid parameter, which avoids cluttering the user's schema.

For the SET and GET procedures, if stattab is not provided (that is, NULL), then the operation works directly on the dictionary statistics; therefore, you do not need to create these statistics tables if they only plan to modify the dictionary directly. However, if stattab is not NULL, then the SET or GET operation works on the specified user statistics table, and not the dictionary.

When a DBMS_STATS subprogram modifies or deletes the statistics for an object, all the dependent cursors are invalidated by default and corresponding statements are subject to recompilation next time so that the new statistics have immediate effects. This behavior can be altered with the no_invalidate argument.

User-Defined Statistics

DBMS_STATS supports operations on user-defined statistics. When a domain index or column is associated with a statistics type (using the associate statement), operations on the index or column manipulate user-defined statistics. For example, gathering statistics for a domain index (for which an association with a statistics type exists) using the GATHER_INDEX_STATS interface invokes the user-defined statistics collection method of the associated statistics type. Similarly, delete, transfer, import, and export operations manipulate user-defined statistics.

SET and GET operations for user-defined statistics are also supported using a special version of the SET and GET interfaces for columns and indexes.

The following procedures in this package commit the current transaction, perform the operation, and then commit again:

- SET_*
- DELETE_*
- EXPORT_*
- IMPORT_*
- GATHER_*
- *_STAT_TABLE

Types

Types for the minimum and maximum values and histogram endpoints include:

```
TYPE numarray  IS VARRAY(256) OF NUMBER;
TYPE datearray IS VARRAY(256) OF DATE;
TYPE chararray IS VARRAY(256) OF VARCHAR2(4000);
TYPE rawarray  IS VARRAY(256) OF RAW(2000);

type StatRec is record (
  epc    NUMBER,
  minval RAW(2000),
  maxval RAW(2000),
  bkvals NUMARRAY,
  novals NUMARRAY);
```

Types for listing stale tables include:

```
type ObjectElem is record (
  ownname     VARCHAR2(30),     -- owner
  objtype     VARCHAR2(6),      -- 'TABLE' or 'INDEX'
  objname     VARCHAR2(30),     -- table/index
  partname    VARCHAR2(30),     -- partition
  subpartname VARCHAR2(30),     -- subpartition
  confidence  NUMBER);          -- not used
type ObjectTab is TABLE of ObjectElem;
```

Use the following constant to indicate that auto-sample size algorithms should be used:

```
AUTO_SAMPLE_SIZE CONSTANT NUMBER;
```

The constant used to determine the system default degree of parallelism, based on the initialization parameters, is:

```
DEFAULT_DEGREE CONSTANT NUMBER;
```

Setting or Getting Statistics

Use the following procedures to store and retrieve individual column-related, index-related, and table-related statistics:

```
PREPARE_COLUMN_VALUES
SET_COLUMN_STATS
SET_INDEX_STATS
SET_SYSTEM_STATS
SET_TABLE_STATS
```

In the special versions of the SET_*_STATS procedures for setting user-defined statistics, the following, if provided, are stored in the dictionary or external statistics table:

- User-defined statistics (extstats)
- The statistics type schema name (statsschema)
- The statistics type name (statsname)

The user-defined statistics and the corresponding statistics type are inserted into the USTATS$ dictionary table. You can specify user-defined statistics without specifying the statistics type name.

```
CONVERT_RAW_VALUE
GET_COLUMN_STATS
GET_INDEX_STATS
GET_SYSTEM_STATS
GET_TABLE_STATS
```

The special versions of the GET_*_STATS procedures return user-defined statistics and the statistics type owner and name as OUT arguments corresponding to the schema object specified. If user-defined statistics are not collected, NULL values are returned.

```
DELETE_COLUMN_STATS
DELETE_INDEX_STATS
DELETE_SYSTEM_STATS
DELETE_TABLE_STATS
DELETE_SCHEMA_STATS
```

```
DELETE_DATABASE_STATS
```

The DELETE_* procedures delete user-defined statistics and the standard statistics for the given schema object.

Transferring Statistics

Use the following procedures to transfer statistics from the dictionary to a user stat table (export_*) and from a user stat table to the dictionary (import_*):

```
CREATE_STAT_TABLE
DROP_STAT_TABLE
```

CREATE_STAT_TABLE can hold user-defined statistics and the statistics type object number.

```
EXPORT_COLUMN_STATS
EXPORT_INDEX_STATS
EXPORT_SYSTEM_STATS
EXPORT_TABLE_STATS
EXPORT_SCHEMA_STATS
EXPORT_DATABASE_STATS

IMPORT_COLUMN_STATS
IMPORT_INDEX_STATS
IMPORT_SYSTEM_STATS
IMPORT_TABLE_STATS
IMPORT_SCHEMA_STATS
IMPORT_DATABASE_STATS
```

The IMPORT_* procedures retrieve statistics, including user-defined statistics, from the stattab table and store them in the dictionary. Because the SET_*_STATS and GET_*_STATS interfaces are supported for user-defined statistics, user-defined statistics can be copied to another database using this interface.

Gathering Optimizer Statistics

Use the following procedures to gather certain classes of optimizer statistics, with possible performance improvements over the ANALYZE command:

```
GATHER_INDEX_STATS
GATHER_TABLE_STATS
GATHER_SCHEMA_STATS
GATHER_DATABASE_STATS
```

GATHER_SYSTEM_STATS

The GATHER_* procedures also collects user-defined statistics for columns and domain indexes.

The statown, stattab, and statid parameters instruct the package to back up current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating statistics for derived objects when you have sufficient statistics on related objects:

GENERATE_STATS

Summary of DBMS_STATS Subprograms

Table 70–1 DBMS_STATS Subprograms

Subprogram	Description
PREPARE_COLUMN_VALUES Procedure on page 70-9	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET_COLUMN_STATS.
SET_COLUMN_STATS Procedure on page 70-11	Sets column-related information.
SET_INDEX_STATS Procedure on page 70-14	Sets index-related information.
SET_SYSTEM_STATS Procedure on page 70-16	Sets system statistics.
SET_TABLE_STATS Procedure on page 70-18	Sets table-related information.
CONVERT_RAW_VALUE Procedure on page 70-19	Convert the internal representation of a minimum or maximum value into a datatype-specific value.
GET_COLUMN_STATS Procedure on page 70-20	Gets all column-related information.
GET_INDEX_STATS Procedure on page 70-22	Gets all index-related information.
GET_SYSTEM_STATS Procedure on page 70-24	Gets system statistics from stattab, or from the dictionary if stattab is null.
GET_TABLE_STATS Procedure on page 70-26	Gets all table-related information.

Table 70–1 (Cont.) DBMS_STATS Subprograms

Subprogram	Description
DELETE_COLUMN_STATS Procedure on page 70-27	Deletes column-related statistics.
DELETE_INDEX_STATS Procedure on page 70-28	Deletes index-related statistics.
DELETE_SYSTEM_STATS Procedure on page 70-29	Deletes system statistics.
DELETE_TABLE_STATS Procedure on page 70-30	Deletes table-related statistics.
DELETE_SCHEMA_STATS Procedure on page 70-31	Deletes schema-related statistics.
DELETE_DATABASE_STATS Procedure on page 70-32	Deletes statistics for the entire database.
CREATE_STAT_TABLE Procedure on page 70-33	Creates a table with name `stattab` in ownname's schema which is capable of holding statistics.
DROP_STAT_TABLE Procedure on page 70-34	Drops a user stat table created by CREATE_STAT_TABLE.
EXPORT_COLUMN_STATS Procedure on page 70-35	Retrieves statistics for a particular column and stores them in the user stat table identified by `stattab`.
EXPORT_INDEX_STATS Procedure on page 70-36	Retrieves statistics for a particular index and stores them in the user stat table identified by `stattab`.
EXPORT_SYSTEM_STATS Procedure on page 70-36	Retrieves system statistics and stores them in the user stat table.
EXPORT_TABLE_STATS Procedure on page 70-37	Retrieves statistics for a particular table and stores them in the user stat table.
EXPORT_SCHEMA_STATS Procedure on page 70-38	Retrieves statistics for all objects in the schema identified by ownname and stores them in the user stat table identified by `stattab`.
EXPORT_DATABASE_STATS Procedure on page 70-39	Retrieves statistics for all objects in the database and stores them in the user stat table identified by `statown.stattab`.
IMPORT_COLUMN_STATS Procedure on page 70-40	Retrieves statistics for a particular column from the user stat table identified by `stattab` and stores them in the dictionary.

Table 70–1 (Cont.) DBMS_STATS Subprograms

Subprogram	Description
IMPORT_INDEX_STATS Procedure on page 70-41	Retrieves statistics for a particular index from the user stat table identified by `stattab` and stores them in the dictionary.
IMPORT_SYSTEM_STATS Procedure on page 70-42	Retrieves system statistics from the user stat table and stores them in the dictionary
IMPORT_TABLE_STATS Procedure on page 70-43	Retrieves statistics for a particular table from the user stat table identified by `stattab` and stores them in the dictionary.
IMPORT_SCHEMA_STATS Procedure on page 70-44	Retrieves statistics for all objects in the schema identified by `ownname` from the user stat table and stores them in the dictionary.
IMPORT_DATABASE_STATS Procedure on page 70-45	Retrieves statistics for all objects in the database from the user stat table and stores them in the dictionary.
GATHER_INDEX_STATS Procedure on page 70-45	Gathers index statistics.
GATHER_TABLE_STATS Procedure on page 70-47	Gathers table and column (and index) statistics.
GATHER_SCHEMA_STATS Procedure on page 70-49	Gathers statistics for all objects in a schema.
GATHER_DATABASE_STATS Procedure on page 70-53	Gathers statistics for all objects in the database.
GATHER_SYSTEM_STATS Procedure on page 70-57	Gathers system statistics.
GENERATE_STATS Procedure on page 70-58	Generates object statistics from previously collected statistics of related objects.
FLUSH_SCHEMA_ MONITORING_INFO Procedure on page 70-59	Flushes in-memory monitoring information for the tables in the specified schema in the dictionary.
FLUSH_DATABASE_ MONITORING_INFO Procedure on page 70-60	Flushes in-memory monitoring information for all the tables to the dictionary.
ALTER_SCHEMA_TABLE_ MONITORING Procedure on page 70-61	Enables or disables the DML monitoring feature of all tables in the schema, except for snapshot logs and the tables, which monitoring does not support.

Table 70–1 (Cont.) DBMS_STATS Subprograms

Subprogram	Description
ALTER_DATABASE_TABLE_MONITORING Procedure on page 70-61	Enables or disables the DML monitoring feature of all tables in the database, except for snapshot logs and the tables, which monitoring does not support.

PREPARE_COLUMN_VALUES Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET_COLUMN_STATS.

Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   charvals        CHARARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec      IN OUT StatRec,
   datevals         DATEARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   numvals         NUMARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   rawvals         RAWARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES_NVARCHAR (
   srec  IN OUT StatRec,
   nvmin        NVARCHAR2,
   nvmax        NVARCHAR2);

DBMS_STATS.PREPARE_COLUMN_VALUES_ROWID (
   srec  IN OUT StatRec,
   rwmin        ROWID,
   rwmax        ROWID);
```

Pragmas

```
pragma restrict_references(prepare_column_values, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 70–2 PREPARE_COLUMN_VALUES Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in Table 70–3.

Table 70–3 Datatype-Specific Input Parameters

Type	Description
charvals	The array of values when the column type is character-based. Up to the first 32 bytes of each string should be provided. Arrays must have between 2 and 256 entries, inclusive. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.
datevals	The array of values when the column type is date-based.

Table 70–3 Datatype-Specific Input Parameters

Type	Description
numvals	The array of values when the column type is numeric-based.
rawvals	The array of values when the column type is RAW. Up to the first 32 bytes of each strings should be provided.
nvmin, nvmax	The minimum and maximum values when the column type is national character set based (NLS). No histogram information can be provided for a column of this type. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.
rwmin, rwmax	The minimum and maximum values when the column type is rowid. No histogram information is provided for a column of this type.

Output Parameters

Table 70–4 PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS.
srec.maxval	Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS.
srec.bkvals	Array suitable for use in a call to SET_COLUMN_STATS.
srec.novals	Array suitable for use in a call to SET_COLUMN_STATS.

Exceptions

ORA-20001: Invalid or inconsistent input values.

SET_COLUMN_STATS Procedure

This procedure sets column-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store

in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

Syntax

Use the following for standard statistics:

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   distcnt       NUMBER DEFAULT NULL,
   density       NUMBER DEFAULT NULL,
   nullcnt       NUMBER DEFAULT NULL,
   srec          StatRec DEFAULT NULL,
   avgclen       NUMBER DEFAULT NULL,
   flags         NUMBER DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   ext_stats     RAW,
   stattypown    VARCHAR2 DEFAULT NULL,
   stattypname   VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–5 SET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.

Table 70–5 SET_COLUMN_STATS Procedure Parameters

Parameter	Description
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
ext_stats	The user-defined statistics.
stattypown	Schema of the statistics type.
stattypname	Name of the statistics type.
distcnt	Number of distinct values.
density	Column density. If this value is NULL and if distcnt is not NULL, then density is derived from distcnt.
nullcnt	Number of NULLs.
srec	StatRec structure filled in by a call to PREPARE_COLUMN_VALUES or GET_COLUMN_STATS.
avgclen	Average length for the column (in bytes).
flags	For internal Oracle use (should be left as NULL).
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent input values.

SET_INDEX_STATS Procedure

This procedure sets index-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

Syntax

Use the following for standard statistics:

```
DBMS_STATS.SET_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2  DEFAULT NULL,
   stattab       VARCHAR2  DEFAULT NULL,
   statid        VARCHAR2  DEFAULT NULL,
   numrows       NUMBER    DEFAULT NULL,
   numlblks      NUMBER    DEFAULT NULL,
   numdist       NUMBER    DEFAULT NULL,
   avglblk       NUMBER    DEFAULT NULL,
   avgdblk       NUMBER    DEFAULT NULL,
   clstfct       NUMBER    DEFAULT NULL,
   indlevel      NUMBER    DEFAULT NULL,
   flags         NUMBER    DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE,
   guessq NUMBER DEFAULT NULL);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2  DEFAULT NULL,
   stattab       VARCHAR2  DEFAULT NULL,
   statid        VARCHAR2  DEFAULT NULL,
   ext_stats     RAW,
   stattypown    VARCHAR2 DEFAULT NULL,
   stattypname   VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE,
```

Parameters

Table 70–6 SET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition in which to store the statistics. If the index is partitioned and if partname is NULL, then the statistics are stored at the global index level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
ext_stats	The user-defined statistics.
stattypown	Schema of the statistics type.
stattypname	Name of the statistics type.
numrows	Number of rows in the index (partition).
numlblks	Number of leaf blocks in the index (partition).
numdist	Number of distinct keys in the index (partition).
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition). If not provided, then this value is derived from numlblks and numdist.
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition). If not provided, then this value is derived from clstfct and numdist.
clstfct	See clustering_factor column of the all_indexes view for a description.
indlevel	Height of the index (partition).
flags	For internal Oracle use (should be left as NULL).
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Table 70–6 SET_INDEX_STATS Procedure Parameters

Parameter	Description
guessq	Guess quality. See the pct_direct_access column of the all_indexes view for a description.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

SET_SYSTEM_STATS Procedure

This procedure sets systems statistics.

Syntax

```
DBMS_STATS.SET_SYSTEM_STATS (
   pname          VARCHAR2,
   pvalue         NUMBER,
   stattab   IN   VARCHAR2 DEFAULT NULL,
   statid    IN   VARCHAR2 DEFAULT NULL,
   statown   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–7 SET_SYSTEM_STATS Procedure Parameters

Parameter	Description
pname	The parameter name to get, which can have one of the following values: ▪ sreadtim—average time to read single block (random read), in milliseconds ▪ mreadtim—average time to read an mbrc block at once (sequential read), in milliseconds ▪ cpuspeed—average number of CPU cycles per second, in millions ▪ mbrc—average multiblock read count for sequential read, in blocks ▪ maxthr—maximum I/O system throughput, in bytes/sec ▪ slavethr—average slave I/O throughput, in bytes/sec
pvalue	Parameter value to get.
stattab	Identifier of the user stat table where the statistics will be obtained. If stattab is null, the statistics will be obtained from the dictionary.
statid	Optional identifier associated with the statistics saved in the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to set system statistics.

ORA-20004: Parameter does not exist.

SET_TABLE_STATS Procedure

This procedure sets table-related information.

Syntax

```
DBMS_STATS.SET_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2 DEFAULT NULL,
   statid   VARCHAR2 DEFAULT NULL,
   numrows  NUMBER   DEFAULT NULL,
   numblks  NUMBER   DEFAULT NULL,
   avgrlen  NUMBER   DEFAULT NULL,
   flags    NUMBER   DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–8 SET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the table (partition).
numblks	Number of blocks the table (partition) occupies.
avgrlen	Average row length for the table (partition).
flags	For internal Oracle use (should be left as NULL).

Table 70–8 (Cont.) SET_TABLE_STATS Procedure Parameters

Parameter	Description
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

CONVERT_RAW_VALUE Procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the StatRec structure as filled in by GET_COLUMN_STATS or PREPARE_COLUMN_VALUES are appropriate values for input.

Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT VARCHAR2);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT DATE);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT NUMBER);

DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR (
   rawval     RAW,
   resval OUT NVARCHAR2);

DBMS_STATS.CONVERT_RAW_VALUE_ROWID (
   rawval     RAW,
   resval OUT ROWID);
```

Pragmas

```
pragma restrict_references(convert_raw_value, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(convert_raw_value_nvarchar, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(convert_raw_value_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 70–9 CONVERT_RAW_VALUE Procedure Parameters

Parameter	Description
rawval	The raw representation of a column minimum or maximum datatype-specific output parameters.
resval	The converted, type-specific value.

GET_COLUMN_STATS Procedure

This procedure gets all column-related information. In the version of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

Use the following for standard statistics:

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   colname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   distcnt OUT NUMBER,
   density OUT NUMBER,
   nullcnt OUT NUMBER,
   srec    OUT StatRec,
   avgclen OUT NUMBER,
   statown     VARCHAR2 DEFAULT NULL);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname          VARCHAR2,
```

```
   tabname          VARCHAR2,
   colname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   ext_stats   OUT  RAW,
   stattypown  OUT  VARCHAR2 DEFAULT NULL,
   stattypname OUT  VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–10 GET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
ext_stats	The user-defined statistics.
stattypown	Schema of the statistics type.
stattypname	Name of the statistics type.
distcnt	Number of distinct values.
density	Column density.
nullcnt	Number of NULLs.
srec	Structure holding internal representation of column minimum, maximum, and histogram values.
avgclen	Average length of the column (in bytes).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_INDEX_STATS Procedure

This procedure gets all index-related information. In the version of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

Use the following for standard statistics:

```
DBMS_STATS.GET_INDEX_STATS (
   ownname      VARCHAR2,
   indname      VARCHAR2,
   partname     VARCHAR2 DEFAULT NULL,
   stattab      VARCHAR2 DEFAULT NULL,
   statid       VARCHAR2 DEFAULT NULL,
   numrows  OUT NUMBER,
   numlblks OUT NUMBER,
   numdist  OUT NUMBER,
   avglblk  OUT NUMBER,
   avgdblk  OUT NUMBER,
   clstfct  OUT NUMBER,
   indlevel OUT NUMBER,
   statown      VARCHAR2 DEFAULT NULL);

DBMS_STATS.GET_INDEX_STATS (
   ownname      VARCHAR2,
   indname      VARCHAR2,
   partname     VARCHAR2 DEFAULT NULL,
   stattab      VARCHAR2 DEFAULT NULL,
   statid       VARCHAR2 DEFAULT NULL,
   numrows  OUT NUMBER,
   numlblks OUT NUMBER,
   numdist  OUT NUMBER,
   avglblk  OUT NUMBER,
   avgdblk  OUT NUMBER,
   clstfct  OUT NUMBER,
   indlevel OUT NUMBER,
   statown      VARCHAR2 DEFAULT NULL,
```

```
   guessq   OUT NUMBER);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   ext_stats   OUT  RAW,
   stattypown  OUT  VARCHAR2 DEFAULT NULL,
   stattypname OUT  VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
```

Parameters

Table 70–11 GET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition for which to get the statistics. If the index is partitioned and if partname is NULL, then the statistics are retrieved for the global index level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
ext_stats	The user-defined statistics.
stattypown	Schema of the statistics type.
stattypname	Name of the statistics type.
numrows	Number of rows in the index (partition).
numlblks	Number of leaf blocks in the index (partition).
numdist	Number of distinct keys in the index (partition).
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition).

Table 70–11 (Cont.) GET_INDEX_STATS Procedure Parameters

Parameter	Description
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition).
clstfct	Clustering factor for the index (partition).
indlevel	Height of the index (partition).
statown	Schema containing stattab (if different than ownname).
guessq	Guess quality for the index (partition).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_SYSTEM_STATS Procedure

This procedure gets system statistics from stattab, or from the dictionary if stattab is null.

Syntax

```
DBMS_STATS.GET_SYSTEM_STATS (
   status    OUT  VARCHAR2,
   dstart    OUT  DATE,
   dstop     OUT  DATE,
   pname          VARCHAR2,
   pvalue    OUT  NUMBER,
   stattab   IN   VARCHAR2 DEFAULT NULL,
   statid    IN   VARCHAR2 DEFAULT NULL,
   statown   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–12 GET_SYSTEM_STATS Procedure Parameters

Parameter	Description
status (OUT)	Output is one of the following: COMPLETED: AUTOGATHERING: MANUALGATHERING: BADSTATS:
dstart (OUT)	Date when statistics gathering started. If status = MANUALGATHERING, the start date is returned.
dstop (OUT)	Date when statistics gathering stopped. If status = COMPLETE, the finish date is returned. If status = AUTOGATHERING, the future finish date is returned. If status = BADSTATS, the must-finished-by date is returned.
pname	The parameter name to get, which can have one of the following values: ▪ sreadtim—average time to read single block (random read), in milliseconds ▪ mreadtim—average time to read an mbrc block at once (sequential read), in milliseconds ▪ cpuspeed—average number of CPU cycles per second, in millions ▪ mbrc—average multiblock read count for sequential read, in blocks ▪ maxthr—maximum I/O system throughput, in bytes/sec ▪ slavethr—average slave I/O throughput, in bytes/sec
pvalue	The parameter value to get.
stattab	Identifier of the user stat table where the statistics will be obtained. If stattab is null, the statistics will be obtained from the dictionary.
statid	Optional identifier associated with the statistics saved in the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to gather system statistics.

ORA-20004: Parameter does not exist.

GET_TABLE_STATS Procedure

This procedure gets all table-related information.

Syntax

```
DBMS_STATS.GET_TABLE_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   numrows OUT NUMBER,
   numblks OUT NUMBER,
   avgrlen OUT NUMBER,
   statown     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–13 GET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).

Table 70–13 (Cont.) GET_TABLE_STATS Procedure Parameters

Parameter	Description
numrows	Number of rows in the table (partition).
numblks	Number of blocks the table (partition) occupies.
avgrlen	Average row length for the table (partition).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

DELETE_COLUMN_STATS Procedure

This procedure deletes column-related statistics.

Syntax

```
DBMS_STATS.DELETE_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   cascade_parts BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–14 DELETE_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.

Table 70–14 (Cont.) DELETE_COLUMN_STATS Procedure Parameters

Parameter	Description
partname	Name of the table partition for which to delete the statistics. If the table is partitioned and if partname is NULL, then global column statistics are deleted.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to true causes the deletion of statistics for this column for all underlying partitions as well.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_INDEX_STATS Procedure

This procedure deletes index-related statistics.

Syntax

```
DBMS_STATS.DELETE_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   cascade_parts BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–15 DELETE_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition for which to delete the statistics. If the index is partitioned and if partname is NULL, then index statistics are deleted at the global level.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the index is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this index for all underlying partitions as well.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

DELETE_SYSTEM_STATS Procedure

This procedure deletes system statistics.

Syntax

```
DBMS_STATS.DELETE_INDEX_STATS (
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–16 DELETE_INDEX_STATS Procedure Parameters

Parameter	Description
stattab	Identifier of the user stat table where the statistics will be saved.
statid	Optional identifier associated with the statistics saved in the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table; may need to be upgraded.

DELETE_TABLE_STATS Procedure

This procedure deletes table-related statistics.

Syntax

```
DBMS_STATS.DELETE_TABLE_STATS (
   ownname         VARCHAR2,
   tabname         VARCHAR2,
   partname        VARCHAR2  DEFAULT NULL,
   stattab         VARCHAR2  DEFAULT NULL,
   statid          VARCHAR2  DEFAULT NULL,
   cascade_parts   BOOLEAN   DEFAULT TRUE,
   cascade_columns BOOLEAN   DEFAULT TRUE,
   cascade_indexes BOOLEAN   DEFAULT TRUE,
   statown         VARCHAR2 DEFAULT NULL,
   no_invalidate   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–17 DELETE_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this table for all underlying partitions as well.
cascade_columns	Indicates that DELETE_COLUMN_STATS should be called for all underlying columns (passing the cascade_parts parameter).
cascade_indexes	Indicates that DELETE_INDEX_STATS should be called for all underlying indexes (passing the cascade_parts parameter).
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

DELETE_SCHEMA_STATS Procedure

This procedure deletes statistics for an entire schema.

Syntax

```
DBMS_STATS.DELETE_SCHEMA_STATS (
```

```
   ownname       VARCHAR2,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–18 DELETE_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_DATABASE_STATS Procedure

This procedure deletes statistics for an entire database.

Syntax

```
DBMS_STATS.DELETE_DATABASE_STATS (
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–19 DELETE_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
statown	Schema containing stattab. If stattab is not NULL and if statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

CREATE_STAT_TABLE Procedure

This procedure creates a table with name stattab in ownname's schema which is capable of holding statistics. The columns and types that compose this table are not relevant as it should be accessed solely through the procedures in this package.

Syntax

```
DBMS_STATS.CREATE_STAT_TABLE (
   ownname  VARCHAR2,
   stattab  VARCHAR2,
   tblspace VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–20 CREATE_STAT_TABLE Procedure Parameters

Parameter	Description
ownname	Name of the schema.

Table 70–20 (Cont.) CREATE_STAT_TABLE Procedure Parameters

Parameter	Description
stattab	Name of the table to create. This value should be passed as the stattab parameter to other procedures when the user does not want to modify the dictionary statistics directly.
tblspace	Tablespace in which to create the stat tables. If none is specified, then they are created in the user's default tablespace.

Exceptions

ORA-20000: Table already exists or insufficient privileges.

ORA-20001: Tablespace does not exist.

DROP_STAT_TABLE Procedure

This procedure drops a user stat table.

Syntax

```
DBMS_STATS.DROP_STAT_TABLE (
   ownname VARCHAR2,
   stattab VARCHAR2);
```

Parameters

Table 70–21 DROP_STAT_TABLE Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier.

Exceptions

ORA-20000: Table does not exists or insufficient privileges.

EXPORT_COLUMN_STATS Procedure

This procedure retrieves statistics for a particular column and stores them in the user stat table identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–22 EXPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
`ownname`	Name of the schema.
`tabname`	Name of the table to which this column belongs.
`colname`	Name of the column.
`partname`	Name of the table partition. If the table is partitioned and if `partname` is `NULL`, then global and partition column statistics are exported.
`stattab`	User stat table identifier describing where to store the statistics.
`statid`	Identifier (optional) to associate with these statistics within `stattab`.
`statown`	Schema containing `stattab` (if different than `ownname`).

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_INDEX_STATS Procedure

This procedure retrieves statistics for a particular index and stores them in the user stat table identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–23 EXPORT_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are exported.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_SYSTEM_STATS Procedure

This procedure retrieves system statistics and stores them in the user stat table, identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_SYSTEM_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–24 EXPORT_SYSTEM_STATS Procedure Parameters

Parameter	Description
stattab	Identifier of the user stat table that describes where the statistics will be stored.
statid	Optional identifier associated with the statistics stored from the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to export system statistics.

EXPORT_TABLE_STATS Procedure

This procedure retrieves statistics for a particular table and stores them in the user stat table. Cascade results in all index and column stats associated with the specified table being exported as well.

Syntax

```
DBMS_STATS.EXPORT_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   cascade  BOOLEAN  DEFAULT TRUE,
```

```
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–25 EXPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
cascade	If true, then column and index statistics for this table are also exported.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_SCHEMA_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by ownname and stores them in the user stat tables identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_SCHEMA_STATS (
   ownname VARCHAR2,
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–26 EXPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_DATABASE_STATS Procedure

This procedure retrieves statistics for all objects in the database and stores them in the user stat tables identified by statown.stattab

Syntax

```
DBMS_STATS.EXPORT_DATABASE_STATS (
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–27 EXPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing where to store the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab. If statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

IMPORT_COLUMN_STATS Procedure

This procedure retrieves statistics for a particular column from the user stat table identified by stattab and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–28 IMPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition column statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_INDEX_STATS Procedure

This procedure retrieves statistics for a particular index from the user stat table identified by stattab and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–29 IMPORT_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_SYSTEM_STATS Procedure

This procedure retrieves system statistics from the user stat table, identified by stattab, and stores the statistics in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_SYSTEM_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–30 IMPORT_SYSTEM_STATS Procedure Parameters

Parameter	Description
stattab	Identifier of the user stat table where the statistics will be retrieved.
statid	Optional identifier associated with the statistics retrieved from the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to import system statistics.

IMPORT_TABLE_STATS Procedure

This procedure retrieves statistics for a particular table from the user stat table identified by `stattab` and stores them in the dictionary. Cascade results in all index and column stats associated with the specified table being imported as well.

Syntax

```
DBMS_STATS.IMPORT_TABLE_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   cascade       BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–31 IMPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
cascade	If true, then column and index statistics for this table are also imported.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_SCHEMA_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by ownname from the user stat table and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_SCHEMA_STATS (
   ownname       VARCHAR2,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–32 IMPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_DATABASE_STATS Procedure

This procedure retrieves statistics for all objects in the database from the user stat table(s) and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_DATABASE_STATS (
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–33 IMPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab. If statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

GATHER_INDEX_STATS Procedure

This procedure gathers index statistics. It attempts to parallelize as much of the work as possible. Restrictions are described in the individual parameters. This operation will not parallelize with certain types of indexes, including cluster

indexes, domain indexes, and bitmap join indexes. The granularity and no_invalidate arguments are not relevant to these types of indexes.

Syntax

```
DBMS_STATS.GATHER_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   no_invalidate    BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 70–34 GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Schema of index to analyze.
indname	Name of index.
partname	Name of partition.
estimate_percent	Percentage of rows to estimate (NULL means compute). The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the best sample size for good statistics.
stattab	User stat table identifier describing where to save the current statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).
degree	Degree of parallelism (NULL means use of table default value that was specified by the DEGREE clause in the CREATE/ALTER INDEX statement). Use the constant DBMS_STATS.DEFAULT_DEGREE for the default value based on the initialization parameters.

Table 70–34 (Cont.) GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
granularity	The granularity of statistics to collect (only pertinent if the index is partitioned): 'DEFAULT' - gathers global and partition-level statistics 'SUBPARTITION' - gathers subpartition-level statistics 'PARTITION '- gathers partition-level statistics 'GLOBAL' - gathers global statistics 'ALL' - gathers all (subpartition, partition, and global) statistics
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE.

Exceptions

ORA-20000: Index does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_TABLE_STATS Procedure

This procedure gathers table and column (and index) statistics. It attempts to parallelize as much of the work as possible, but there are some restrictions as described in the individual parameters. This operation does not parallelize if the user does not have select privilege on the table being analyzed.

Syntax

```
DBMS_STATS.GATHER_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
```

```
statown          VARCHAR2 DEFAULT NULL,
no_invalidate    BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 70–35 GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of table to analyze.
tabname	Name of table.
partname	Name of partition.
estimate_percent	Percentage of rows to estimate (NULL means compute) The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the best sample size for good statistics.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Accepts: FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...], where size_clause is defined as: size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} integer—Number of histogram buckets. Must be in the range [1,254]. REPEAT—Collects histograms only on the columns that already have histograms. AUTO—Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. SKEWONLY—Oracle determines the columns to collect histograms based on the data distribution of the columns.

Table 70–35 (Cont.) GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
degree	Degree of parallelism. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.
cascade	Gather statistics on the indexes for this table. Index statistics gathering is not parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS procedure on each of the table's indexes.
stattab	User stat table identifier describing where to save the current statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE. When the 'cascade' argument is specified, this parameter is not relevant with certain types of indexes, as described in "GATHER_INDEX_STATS Procedure" on page 70-45.

Exceptions

ORA-20000: Table does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_SCHEMA_STATS Procedure

This procedure gathers statistics for all objects in a schema.

Syntax

```
DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist      OUT ObjectTab,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN DEFAULT FALSE,
   gather_temp      BOOLEAN DEFAULT FALSE);

DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT FALSE,
   gather_temp      BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 70–36 GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Schema to analyze (NULL means current schema).
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the best sample size for good statistics.

Table 70–36 (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Accepts: FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...], where size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} integer—Number of histogram buckets. Must be in the range [1,254]. REPEAT—Collects histograms only on the columns that already have histograms. AUTO—Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. SKEWONLY—Oracle determines the columns to collect histograms based on the data distribution of the columns.
degree	Degree of parallelism. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.

Table 70–36 (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
cascade	Gather statistics on the indexes as well. Index statistics gathering is not parallelized. Using this option is equivalent to running the gather_index_stats procedure on each of the indexes in the schema in addition to gathering table and column statistics.
stattab	User stat table identifier describing where to save the current statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
options	Further specification of which objects to gather statistics for: GATHER: Gathers statistics on all objects in the schema. GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are ownname, stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects. GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale. GATHER EMPTY: Gathers statistics on objects which currently have no statistics. also, return a list of objects found to have no statistics. LIST AUTO: Returns a list of objects to be processed with GATHER AUTO. LIST STALE: Returns list of stale objects as determined by looking at the *_tab_modifications views. LIST EMPTY: Returns list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty.
statown	Schema containing stattab (if different than ownname).
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE. When the 'cascade' argument is specified, this parameter is not relevant with certain types of indexes, as described in "GATHER_INDEX_STATS Procedure" on page 70-45.

Table 70–36 (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
gather_temp	Gathers statistics on global temporary tables. The temporary table must be created with the "on commit preserve rows" clause. The statistics being collected are based on the data in the session in which this procedure is run, but shared across all sessions.

Exceptions

ORA-20000: Schema does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_DATABASE_STATS Procedure

This procedure gathers statistics for all objects in the database.

Syntax

```
DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist   OUT    ObjectTab,
   statown          VARCHAR2 DEFAULT NULL,
   gather_sys       BOOLEAN DEFAULT FALSE,
   no_invalidate    BOOLEAN DEFAULT FALSE,
   gather_temp      BOOLEAN DEFAULT FALSE);

DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
```

```
cascade          BOOLEAN  DEFAULT FALSE,
stattab          VARCHAR2 DEFAULT NULL,
statid           VARCHAR2 DEFAULT NULL,
options          VARCHAR2 DEFAULT 'GATHER',
statown          VARCHAR2 DEFAULT NULL,
gather_sys       BOOLEAN DEFAULT FALSE,
no_invalidate    BOOLEAN DEFAULT FALSE,
gather_temp      BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–37 GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the best sample size for good statistics.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Accepts: FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...], where size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} integer—Number of histogram buckets. Must be in the range [1,254]. REPEAT—Collects histograms only on the columns that already have histograms. AUTO—Oracle determines the columns to collect histograms based on data distribution and the workload of the columns. SKEWONLY—Oracle determines the columns to collect histograms based on the data distribution of the columns.

Table 70–37 (Cont.) GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
degree	Degree of parallelism. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.
cascade	Gather statistics on the indexes as well. Index statistics gathering is not parallelized. Using this option is equivalent to running the gather_index_stats procedure on each of the indexes in the database in addition to gathering table and column statistics.
stattab	User stat table identifier describing where to save the current statistics. The statistics table is assumed to reside in the same schema as the object being analyzed, so there must be one such table in each schema to use this option.
statid	Identifier (optional) to associate with these statistics within stattab.

Table 70–37 (Cont.) GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
options	Further specification of which objects to gather statistics for: GATHER: Gathers statistics on all objects in the schema. GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects. GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale. GATHER EMPTY: Gathers statistics on objects which currently have no statistics. Return a list of objects found to have no statistics. LIST AUTO: Returns a list of objects to be processed with GATHER AUTO. LIST STALE: Returns a list of stale objects as determined by looking at the *_tab_modifications views. LIST EMPTY: Returns a list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty.
statown	Schema containing stattab (if different than ownname).
gather_sys	Gathers statistics on the objects owned by the 'SYS' user.
no_invalidate	Dependent cursors are not invalidated if this parameter is set to TRUE. When the 'cascade' option is specified, this parameter is not relevant with certain types of indexes, as described in "GATHER_INDEX_STATS Procedure" on page 70-45.
gather_temp	Gathers statistics on global temporary tables. The temporary table must be created with the "on commit preserve rows" clause. The statistics being collected are based on the data in the session in which this procedure is run, but shared across all sessions.

Exceptions

ORA-20000: Insufficient privileges.

ORA-20001: Bad input value.

GATHER_SYSTEM_STATS Procedure

This procedure gathers system statistics.

Syntax

```
DBMS_STATS.GATHER_SYSTEM_STATS (
   gathering_mode   VARCHAR2 DEFAULT 'NOWORKLOAD',
   interval         INTEGER  DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–38 GATHER_SYSTEM_STATS Procedure Parameters

Parameter	Description
gathering_mode	Mode values are: NOWORKLOAD: No workload is required to capture system activity. Oracle generates system statistics using internal defaults. This mode can be used when suitable workload cannot be submitted (during the development process, for example). For system statistics values to be based on real system activity, use the INTERVAL or START|STOP modes instead. INTERVAL: Captures system activity during a specified interval. This works in combination with the interval parameter. You should provide an interval value in minutes, after which system statistics are created or updated in the dictionary or stattab. You can use gather_system_stats(gathering_mode=>'STOP') to stop gathering earlier when scheduled. START | STOP: Captures system activity during specified start and stop times and refreshes the dictionary or stattab with statistics for the elapsed period. Interval value is ignored.
interval	Time, in minutes, to gather statistics. This parameter applies only when gathering_mode='INTERVAL'.

Table 70–38 (Cont.) GATHER_SYSTEM_STATS Procedure Parameters

Parameter	Description
stattab	Identifier of the user stat table where the statistics will be saved.
statid	Optional identifier associated with the statistics saved in the stattab.
statown	The schema containing stattab, if different from the user's schema.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

ORA-20002: Bad user statistics table; may need to be upgraded.

ORA-20003: Unable to gather system statistics.

ORA-20004: Error in the INTERVAL mode: system parameter job_queue_processes must be >0.

GENERATE_STATS Procedure

This procedure generates object statistics from previously collected statistics of related objects. For fully populated schemas, the gather procedures should be used instead when more accurate statistics are desired.The currently supported objects are b-tree and bitmap indexes.

Syntax

```
DBMS_STATS.GENERATE_STATS (
   ownname   VARCHAR2,
   objname   VARCHAR2,
   organized NUMBER DEFAULT 7);
```

Parameters

Table 70–39 GENERATE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of object.
objname	Name of object.
organized	Amount of ordering associated between the index and its underlying table. A heavily organized index would have consecutive index keys referring to consecutive rows on disk for the table (the same block). A heavily disorganized index would have consecutive keys referencing different table blocks on disk. This parameter is only used for b-tree indexes. The number can be in the range of 0-10, with 0 representing a completely organized index and 10 a completely disorganized one.

Exceptions

ORA-20000: Unsupported object type of object does not exist.

ORA-20001: Invalid option or invalid statistics.

FLUSH_SCHEMA_MONITORING_INFO Procedure

This procedure flushes in-memory monitoring information for the tables in the specified schema to the dictionary.

Syntax

```
DBMS_STATS.FLUSH_SCHEMA_MONITORING_INFO (
   ownname VARCHAR2 DEFAULT NULL);
```

Parameters

Table 70–40 FLUSH_SCHEMA_MONITORING_INFO Procedure Parameters

Parameter	Description
ownname	The name of the schema. (NULL means the current schema.)

Exceptions

ORA-20000: The object does not exist or it contains insufficient privileges.

FLUSH_DATABASE_MONITORING_INFO Procedure

This procedure flushes in-memory monitoring information for all the tables to the dictionary.

Syntax

```
DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO;
```

Exceptions

ORA-20000: Insufficient privileges.

ALTER_SCHEMA_TABLE_MONITORING Procedure

This procedure enable or disables the DML monitoring feature of all the tables in the schema, except for snapshot logs and the tables, which monitoring does not support. Using this procedure is equivalent to issuing ALTER TABLE...MONITORING (or NOMONITORING) individually. You should enable monitoring if you use GATHER_DATABASE_STATS or GATHER_SCHEMA_STATS with the GATHER AUTO or GATHER STALE options.

Syntax

```
DBMS_STATS.ALTER_SCHEMA_TABLE_MONITORING (
   ownname    VARCHAR2 DEFAULT NULL,
   monitoring BOOLEAN DEFAULT TRUE);
```

Parameters

Table 70–41 ALTER_SCHEMA_TABLE_MONITORING Procedure Parameters

Parameter	Description
ownname	The name of the schema. (NULL means the current schema.)
monitoring	Enables monitoring if true, and disables monitoring if false.

Exceptions

ORA-20000: Insufficient privileges.

ALTER_DATABASE_TABLE_MONITORING Procedure

This procedure enables or disables the DML monitoring feature of all the tables in the schema, except for snapshot logs and the tables, which monitoring does not support. Using this procedure is equivalent to issuing ALTER TABLE...MONITORING (or NOMONITORING) individually. You should enable monitoring if you use GATHER_DATABASE_STATS or GATHER_SCHEMA_STATS with the GATHER AUTO or GATHER STALE options.

Syntax

```
DBMS_STATS.ALTER_DATABASE_TABLE_MONITORING (
   monitoring BOOLEAN DEFAULT TRUE,
   sysobjs    BOOLEAN DEFAULT FALSE);
```

Parameters

Table 70–42 ALTER_DATABASE_TABLE_MONITORING Procedure Parameters

Parameter	Description
monitoring	Enables monitoring if true, and disables monitoring if false.
sysobjs	If true, changes monitoring on the dictionary objects.

Exceptions

ORA-20000: Insufficient privileges.

Saving Original Statistics and Gathering New Statistics: Example

Assume many modifications have been made to the employees table since the last time statistics were gathered. To ensure that the cost-based optimizer is still picking the best plan, statistics should be gathered once again; however, the user is concerned that new statistics will cause the optimizer to choose bad plans when the current ones are acceptable. The user can do the following:

```
BEGIN
   DBMS_STATS.CREATE_STAT_TABLE ('hr', 'savestats');
```

```
   DBMS_STATS.GATHER_TABLE_STATS ('hr', 'employees', stattab => 'savestats');
END;
```

This operation gathers new statistics on the `employees` table, but first saves the original statistics in a user stat table: `hr.savestats`.

If the user believes that the new statistics are causing the optimizer to generate poor plans, then the original stats can be restored as follows:

```
BEGIN
   DBMS_STATS.DELETE_TABLE_STATS ('hr', 'employees');
   DBMS_STATS.IMPORT_TABLE_STATS ('hr', 'employees', stattab => 'savestats');
END;
```

Gathering Daytime System Statistics: Example

Assume that you want to perform database application processing OLTP transactions during the day and run reports at night.

To collect daytime system statistics, gather statistics for 720 minutes. Store the statistics in the MYSTATS table.

```
BEGIN
   DBMS_STATS.GATHER_SYSTEM_STATS (
      interval => 720,
      stattab  => 'mystats',
      statid   => 'OLTP');
END;
```

To collect nighttime system statistics, gather statistics for 720 minutes. Store the statistics in the MYSTATS table.

```
BEGIN
   DBMS_STATS.GATHER_SYSTEM_STATS (
      interval => 720,
      stattab  => 'mystats',
      statid   => 'OLAP');
END;
```

Update the dictionary with the gathered statistics.

```
VARIABLE   jobno   number;
BEGIN
   DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS
   (''mystats'',''OLTP'');'
   sysdate, 'sysdate + 1');
   COMMIT;
```

```
END;

BEGIN
   DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS
   (''mystats'',''OLAP'');'
   sysdate + 0.5, 'sysdate + 1');
   COMMIT;
END;
```

71

DBMS_STORAGE_MAP

With DBMS_STORAGE_MAP, you can communicate with the Oracle background process FMON to invoke mapping operations that populate mapping views. FMON communicates with operating and storage system vendor-supplied mapping libraries.

This chapter discusses the following topics:

Mapping Terminology

The following terminology and descriptions will help you understand the DBMS_STORAGE_MAP API:

- Mapping libraries

  Mapping libraries help you map the components of I/O processing stack elements. Examples of I/O processing components include files, logical volumes, and storage array I/O targets. The mapping libraries are identified in `filemap.ora`.

- Mapping files

  A mapping file is a mapping structure that describes a file. It provides a set of attributes, including file size, number of extents that the file is composed of, and file type.

- Mapping elements and subelements

  A mapping element is the abstract mapping structure that describes a storage component within the I/O stack. Examples of elements include mirrors, stripes, partitions, raid5, concatenated elements, and disks—structures that are the mapping building blocks. A mapping subelement describes the link between an element and the next elements in the I/O mapping stack

- Mapping file extents

  A mapping file extent describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides. In the case of a raw device or volume, the file is composed of only one file extent component. A mapping file extent is different from Oracle extents.

  See Also:

  - *Oracle9i Database Administrator's Guide* for more information
  - *Oracle9i Database Reference* for V$MAP views, including V$MAP_FILE, V$MAP_ELEMENT, V$MAP_SUBELEMENT, V$MAP_FILE_EXTENT

Summary of DBMS_STORAGE_MAP Subprograms

Table 71–1 DBMS_STORAGE_MAP Package Subprograms

Subprogram	Description
MAP_ELEMENT Function on page 71-4	Builds mapping information for the element identified by `elemname`
MAP_FILE Function on page 71-5	Builds mapping information for the file identified by `filename`
MAP_OBJECT Function on page 71-5	Builds the mapping information for the Oracle object identified by the object name, owner, and type
MAP_ALL Function on page 71-5	Builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements
DROP_ELEMENT Function on page 71-6	Drops the mapping information for the element defined by `elemname`
DROP_FILE Function on page 71-6	Drops the file mapping information defined by `filename`
DROP_ALL Function on page 71-7	Drops all mapping information in the shared memory of the instance
SAVE Function on page 71-7	Saves information needed to regenerate the entire mapping into the data dictionary
RESTORE Function on page 71-7	Loads the entire mapping information from the data dictionary into the shared memory of the instance
LOCK_MAP Procedure on page 71-8	Locks the mapping information in the shared memory of the instance
UNLOCK_MAP Procedure on page 71-8	Unlocks the mapping information in the shared memory of the instance.

MAP_ELEMENT Function

This function builds mapping information for the element identified by `elemname`. It may not obtain the latest mapping information if the element being mapped, or any one of the elements within its I/O stack (if `cascade` is `TRUE`), is owned by a library that must be explicitly synchronized.

Syntax

```
DBMS_STORAGE_MAP.MAP_ELEMENT(
```

```
   elemname          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Table 71–2 MAP_ELEMENT Function Parameters

Parameter	Description
elemname	The element for which mapping information is built.
cascade	If TRUE, all elements within the elemname I/O stack DAG are mapped.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

MAP_FILE Function

This function builds mapping information for the file identified by filename. Use this function if the mapping of one particular file has changed. The Oracle database server does not have to rebuild the entire mapping.

This function may not obtain the latest mapping information if the file being mapped, or any one of the elements within its I/O stack (if cascade is TRUE), is owned by a library that must be explicitly synchronized.

Syntax

```
DBMS_STORAGE_MAP.MAP_FILE(
   filename          IN VARCHAR2,
   filetype          IN VARCHAR2,
   cascade           IN BOOLEAN,
   max_num_fileextent IN NUMBER DEFAULT 100,
   dictionary_update  IN BOOLEAN DEFAULT TRUE);
```

Table 71–3 MAP_FILE Function Parameters

Parameter	Description
filename	The file for which mapping information is built.
filetype	Defines the type of the file to be mapped. It can be "DATAFILE", "SPFILE", "TEMPFILE", "CONTROLFILE", "LOGFILE", or "ARCHIVEFILE".

Table 71–3 MAP_FILE Function Parameters

Parameter	Description
cascade	Should be TRUE only if a storage reconfiguration occurred. For all other instances, such as file resizing (either through an ALTER SYSTEM command or DML operations on extended files), cascade can be set to FALSE because the mapping changes are limited to the file extents only. If TRUE, mapping DAGs are also built for the elements where the file resides.
max_num_fileextent	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; max_num_fileextent is an overloaded argument.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

MAP_OBJECT Function

This function builds the mapping information for the Oracle object identified by the object name, owner, and type.

Syntax

```
DBMS_STORAGE_MAP.MAP_OBJECT(
   objname IN VARCHAR2,
   owner   IN VARCHAR2,
   objtype IN VARCHAR2);
```

MAP_ALL Function

This function builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements. It obtains the latest mapping information because it explicitly synchronizes all mapping libraries. You must explicitly call MAP_ALL in a cold startup scenario.

Syntax

```
DBMS_STORAGE_MAP.MAP_ALL(
   max_num_fileext   IN NUMBER DEFAULT 100,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Table 71–4 MAP_ALL Function Parameters

Parameter	Description
max_num_fileext	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; max_num_fileextent is an overloaded argument.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

DROP_ELEMENT Function

This function drops the mapping information for the element defined by elemname.

Syntax

```
DBMS_STORAGE_MAP.DROP_ELEMENT(
   elemname          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Table 71–5 DROP_ELEMENT Function Parameters

Parameter	Description
elemname	The element for which mapping information is dropped.
cascade	If TRUE, then DROP_ELEMENT is invoked recursively on all elements of the DAG defined by elemname, if possible.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

DROP_FILE Function

This function drops the file mapping information defined by filename.

Syntax

```
DBMS_STORAGE_MAP.DROP_FILE(
   filename          IN VARCHAR2,
   cascade           IN BOOLEAN,
   dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Table 71–6 DROP_FILE Function Parameters

Parameter	Description
filename	The file for which file mapping information is dropped.
cascade	If TRUE, then the mapping DAGs for the elements where the file resides are also dropped, if possible.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

DROP_ALL Function

This function drops all mapping information in the shared memory of the instance.

Syntax

```
DBMS_STORAGE_MAP.DROP_ALL(
      dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Table 71–7 DROP_ALL Function Parameters

Parameter	Description
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

SAVE Function

This function saves information needed to regenerate the entire mapping into the data dictionary.

Syntax

```
DBMS_STORAGE_MAP.SAVE();
```

RESTORE Function

This function loads the entire mapping information from the data dictionary into the shared memory of the instance. You can invoke RESTORE only after a SAVE operation. You must explicitly call RESTORE in a warm startup scenario.

Syntax

```
DBMS_STORAGE_MAP.RESTORE();
```

LOCK_MAP Procedure

This procedure locks the mapping information in the shared memory of the instance. This is useful when you need a consistent snapshot of the V$MAP tables. Without locking the mapping information, V$MAP_ELEMENT and V$MAP_SUBELEMENT, for example, may be inconsistent.

Syntax

```
DBMS_STORAGE_MAP.LOCK_MAP();
```

UNLOCK_MAP Procedure

This procedure unlocks the mapping information in the shared memory of the instance.

Syntax

```
DBMS_STORAGE_MAP.LOCK_MAP();
```

Usage Notes for DBMS_STORAGE_MAP Subprograms

For MAP_ELEMENT, MAP_FILE, and MAP_ALL: Invoking these functions when mapping information already exists will refresh the mapping if configuration IDs are supported. If configuration IDs are not supported, then invoking these functions again will rebuild the mapping.

> **See Also:** *Oracle9i Database Administrator's Guide* for a discussion of the configuration ID, an attribute of the element or file that is changed.

72

DBMS_STREAMS

The DBMS_STREAMS package provides interfaces to convert SYS.AnyData objects into logical change record (LCR) objects, to return information about Streams attributes, and to annotate redo entries generated by a session with a binary tag. This tag affects the behavior of a capture process, a propagation job, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

This chapter contains the following topic:

- Summary of DBMS_STREAMS Subprograms

Note: PUBLIC is granted execute privilege on this package.

See Also: *Oracle9i Streams* for more information about Streams

Summary of DBMS_STREAMS Subprograms

Table 72–1 DBMS_STREAMS Subprograms

Subprogram	Description
"CONVERT_ANYDATA_TO_LCR_DDL Function" on page 72-3	Converts a `SYS.AnyData` object to a `SYS.LCR$_DDL_RECORD` object
"CONVERT_ANYDATA_TO_LCR_ROW Function" on page 72-4	Converts a `SYS.AnyData` object to a `SYS.LCR$_ROW_RECORD` object
"GET_INFORMATION Function" on page 72-5	Returns information about various Streams attributes
"GET_TAG Function" on page 72-6	Gets the binary tag for all redo entries generated by the current session
"SET_TAG Procedure" on page 72-7	Sets the binary tag for all redo entries subsequently generated by the current session

CONVERT_ANYDATA_TO_LCR_DDL Function

Converts a SYS.AnyData object into a SYS.LCR$_DDL_RECORD object. You can specify this function in a rule-based transformation when propagating data definition language (DDL) LCRs from a SYS.AnyData queue to a SYS.LCR$_DDL_RECORD typed queue.

Alternatively, you can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Then, use the transformation you create when you add a subscriber for propagation of DDL LCRs from a SYS.AnyData queue to a SYS.LCR$_DDL_RECORD typed queue.

> **See Also:** *Oracle9i Streams* for more information about this function

Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_DDL(
   source     IN  SYS.AnyData)
RETURN SYS.LCR$_DDL_RECORD;
```

Parameter

Table 72–2 CONVERT_ANYDATA_TO_LCR_DDL Function Parameter

Parameter	Description
source	The SYS.AnyData object to be converted. If this object is not a DDL LCR, then an exception is raised.

CONVERT_ANYDATA_TO_LCR_ROW Function

Converts a SYS.AnyData object into a SYS.LCR$_ROW_RECORD object. You can use this function in a rule-based transformation when propagating row LCRs from a SYS.AnyData queue to a SYS.LCR$_ROW_RECORD typed queue.

Alternatively, you can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Then, use the transformation you create when you add a subscriber for propagation of row LCRs from a SYS.AnyData queue to a SYS.LCR$_ROW_RECORD typed queue.

> **See Also:** *Oracle9i Streams* for more information about this function

Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(
   source       IN  SYS.AnyData)
RETURN SYS.LCR$_ROW_RECORD;
```

Parameter

Table 72–3 CONVERT_ANYDATA_TO_LCR_ROW Function Parameter

Parameter	Description
source	The SYS.AnyData object to be converted. If this object is not a row LCR, then an exception is raised.

GET_INFORMATION Function

Returns information about various Streams attributes.

Syntax

```
DBMS_STREAMS.GET_INFORMATION(
   name      IN  VARCHAR2)
RETURN SYS.AnyData;
```

Parameter

Table 72–4 GET_INFORMATION Function Parameter

Parameter	Description
name	The type of information you want to retrieve. Currently, the following names are available: ■ SENDER: Returns the name of the sender for the current LCR (from its AQ message properties). This function is called inside an apply handler. An apply handler is a DML handler, a DDL handler, an error handler, or a message handler. Returns NULL if called outside of an apply handler. The return value is to be interpreted as a VARCHAR2. ■ CONSTRAINT_NAME: Returns the name of the constraint that was violated for an LCR that raised an error. This function is called inside a DML handler or error handler for an apply process. Returns NULL if called outside of a DML handler or error handler. The return value is to be interpreted as a VARCHAR2.

GET_TAG Function

Gets the binary tag for all redo entries generated by the current session.

See Also: *Oracle9i Streams* for more information about tags

Syntax

```
DBMS_STREAMS.GET_TAG()
RETURN RAW;
```

Usage Notes

The following example illustrates how to display the current LCR tag as output:

```
SET SERVEROUTPUT ON
DECLARE
   raw_tag RAW(2000);
BEGIN
   raw_tag := DBMS_STREAMS.GET_TAG();
   DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the value by querying the DUAL view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

SET_TAG Procedure

Sets the binary tag for all redo entries subsequently generated by the current session. Each redo entry generated by DML or DDL statements in the current session will have this tag. This procedure affects only the current session.

Note: This procedure is not transactional. That is, the effects of SET_TAG cannot be rolled back.

See Also: *Oracle9i Streams* for more information about tags

Syntax

```
DBMS_STREAMS.SET_TAG(
   tag   IN RAW   DEFAULT NULL);
```

Parameter

Table 72–5 SET_TAG Procedure Parameter

Parameter	Description
tag	The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits. By default, the tag for a session is NULL. The size limit for a tag value is 2000 bytes.

Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

```
EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));
```

73

DBMS_STREAMS_ADM

The DBMS_STREAMS_ADM package provides administrative procedures for adding and removing simple rules, without transformations, for capture, propagation, and apply at the table, schema, and database level. These rules support logical change records (LCRs), which include row LCRs and data definition language (DDL) LCRs. This package also contains procedures for creating queues and for managing Streams metadata, such as data dictionary information.

This chapter contains the following topic:

- Summary of DBMS_STREAMS_ADM Subprograms

If you require more sophisticated rules or rules involving non-LCR events, then you can use the DBMS_RULE_ADM package.

If you require transformations on simple rules, then you can use the DBMS_RULE_ADM package to add, update, or remove transformations on rules created by the DBMS_STREAMS_ADM package.

See Also:

- *Oracle9i Streams* for more information about Streams
- Chapter 64, "DBMS_RULE_ADM"

Summary of DBMS_STREAMS_ADM Subprograms

Table 73–1 DBMS_STREAMS_ADM Subprograms

Subprogram	Description
"ADD_GLOBAL_PROPAGATION_RULES Procedure" on page 73-3	Adds propagation rules that propagate all the LCRs in a source queue to a destination queue
"ADD_GLOBAL_RULES Procedure" on page 73-7	Adds capture rules for an entire database or apply rules for all LCRs in a queue
"ADD_SCHEMA_PROPAGATION_RULES Procedure" on page 73-11	Adds propagation rules that propagate the LCRs related to the specified schema in a source queue to a destination queue
"ADD_SCHEMA_RULES Procedure" on page 73-15	Adds capture or apply rules for a schema
"ADD_SUBSET_RULES Procedure" on page 73-19	Adds apply rules for a subset of the rows in a table
"ADD_TABLE_PROPAGATION_RULES Procedure" on page 73-24	Adds propagation rules that propagate the LCRs related to the specified table in a source queue to a destination queue
"ADD_TABLE_RULES Procedure" on page 73-28	Adds capture or apply rules for a table
"PURGE_SOURCE_CATALOG Procedure" on page 73-32	Removes all Streams data dictionary information at the local database for the specified object
"REMOVE_RULE Procedure" on page 73-34	Removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, or propagation job
"SET_UP_QUEUE Procedure" on page 73-35	Creates a queue table and a queue for use with the capture, propagate, and apply functionality of Streams

Note: All procedures commit unless specified otherwise.

ADD_GLOBAL_PROPAGATION_RULES Procedure

Adds propagation rules that propagate all the LCRs in a source queue to a destination queue. This procedure also configures propagation using the current user, if necessary, and establishes a default propagation schedule. This procedure enables propagation of all LCRs in the source queue, subject to filtering conditions, to the destination queue. Only one propagation job is allowed between the source queue and destination queue.

If propagation rules are added, then the propagation job propagates DML changes, or DDL changes, or both from the specified source queue to the specified destination queue. This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. A system-generated rule name is the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. For the overloaded `ADD_GLOBAL_PROPAGATION_RULES` procedure, the system-generated rule names for DML and DDL changes are returned.

A propagation job uses the rules created for filtering. If the propagation job does not have a rule set, then a rule set is created automatically, and the rules for propagating changes to the database are added to the rule set. Other rules in an existing rule set for the propagation job are not affected. You can add additional rules using the `DBMS_RULE_ADM` package.

The following is an example of a global rule condition that may be created for propagating DML changes with a propagation job:

```
:dml.get_source_database_name() = 'DBS1.NET' AND :dml.is_null_tag() = 'Y'
```

Note: The quotation marks in the preceding example are all single quotation marks.

For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate events.

Note:

- Currently, a single propagation job propagates all events that use a particular database link, even if the database link propagates events to multiple destination queues.
- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users may or may not be the same.

See Also: "CREATE_PROPAGATION Procedure" on page 47-4 for more information about the required privileges

Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
   streams_name              IN  VARCHAR2,
   source_queue_name         IN  VARCHAR2,
   destination_queue_name    IN  VARCHAR2,
   include_dml               IN  BOOLEAN   DEFAULT true,
   include_ddl               IN  BOOLEAN   DEFAULT false,
   include_tagged_lcr        IN  BOOLEAN   DEFAULT false,
   source_database           IN  VARCHAR2  DEFAULT NULL,
   dml_rule_name             OUT VARCHAR2,
   ddl_rule_name             OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Parameters

Table 73–2 ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

Parameter	Description
streams_name	The name of the propagation job. If the specified propagation job does not exist, then it is created automatically. If NULL and a propagation job exists for the same source queue and destination queue (including database link), then this propagation job is used. If NULL and no propagation job exists for the same source queue and destination queue (including database link), then a propagation job is created automatically with a system-generated name.
source_queue_name	The name of the source queue. The current database must contain the source queue.
destination_queue_name	The name of the destination queue, including any database link, such as STREAMS_QUEUE@DBS2. If the database link is omitted, then the global name of the current database is used, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then creates a rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags

Table 73–2 ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters (Cont.)

Parameter	Description
source_database	The global name of the source database. The source database is where the changes originated. If NULL, then no condition regarding the source database is added to the generated rules. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically. Oracle Corporation recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

ADD_GLOBAL_RULES Procedure

Adds capture rules for an entire database or apply rules for all LCRs in a queue.

If capture rules are added, then captures DML changes, or DDL changes, or both in the current database and enqueues these changes into the specified queue. For capture rules, you should execute this procedure at the source database. This procedure automatically invokes the PREPARE_GLOBAL_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.

If apply rules are added, then the apply process receives and applies captured events that contain DML changes, or DDL changes, or both that originated at the source database matching the source_database parameter. For apply rules, you should execute this procedure at the destination database.

An apply process created by this procedure can apply events only at the local database and can apply only captured events. To create an apply process that applies events at a remote database or an apply process that applies user-enqueued events, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

Changes applied by an apply process created by this procedure generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

You have the option of creating an apply process using the DBMS_APPLY_ADM.CREATE_APPLY procedure and specifying nondefault values for the apply_captured, apply_database_link, and apply_tag parameters when you run that procedure. Then you can use this ADD_GLOBAL_RULES procedure to add rules to the rule set used by the apply process.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. A system-generated rule name is the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated.

For the overloaded ADD_GLOBAL_RULES procedure, the system-generated rule names for DML and DDL changes are returned.

A capture process or an apply process uses the rules created for filtering. If the generated process does not have a rule set, then a rule set is created automatically, and the rules are added to the rule set. Other rules in an existing rule set for the process are not affected. You can add additional rules using the DBMS_RULE_ADM package.

The following is an example of a global rule condition that may be created for capturing DML changes with a capture process:

```
:dml.is_null_tag() = 'Y'
```

The following is an example of a global rule condition that may be created for applying DML changes with an apply process:

```
:dml.get_source_database_name() = 'DBS1.NET' AND :dml.is_null_tag() = 'Y'
```

Note: The quotation marks in the preceding example are all single quotation marks.

If this procedure creates a capture process or an apply process, then the user who runs this procedure is the user who captures or applies changes. The specified user must have the necessary privileges to perform these actions.

See Also:

- Chapter 64, "DBMS_RULE_ADM"
- "CREATE_CAPTURE Procedure" on page 8-6 for information about the privileges required to capture changes
- "CREATE_APPLY Procedure" on page 4-9 for information about the privileges required to apply changes (refer to the apply_user parameter)

Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
   streams_type             IN  VARCHAR2,
   streams_name             IN  VARCHAR2 DEFAULT NULL,
   queue_name               IN  VARCHAR2 DEFAULT 'streams_queue',
   include_dml              IN  BOOLEAN  DEFAULT true,
   include_ddl              IN  BOOLEAN  DEFAULT false,
   include_tagged_lcr       IN  BOOLEAN  DEFAULT false,
   source_database          IN  VARCHAR2 DEFAULT NULL,
   dml_rule_name           OUT VARCHAR2,
   ddl_rule_name           OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

Parameters

Table 73–3 ADD_GLOBAL_RULES Procedure Parameters **(Page 1 of 2)**

Parameter	Description
streams_type	The type of process, either capture or apply
streams_name	The name of the capture or apply process. If the specified process does not exist, then it is created automatically. If NULL and one relevant capture or apply process for the queue exists, then the relevant process is used. If no relevant process exists for the queue, then a capture process or an apply process is created automatically with a system-generated name. If NULL and multiple processes of the specified streams_type for the queue exist, then an error is raised.
queue_name	The name of the local queue. For capture rules, the queue into which the changes will be enqueued. For apply rules, the queue from which changes will be dequeued.
include_dml	If TRUE, then creates a rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.

Table 73–3 ADD_GLOBAL_RULES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
include_tagged_lcr	If TRUE, then a redo entry is always considered for capture and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then a redo entry is considered for capture and an LCR is considered for apply only when the redo entry or the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags
source_database	The global name of the source database. If NULL, then no condition regarding the source database is added to the generated rules. For capture rules, you can specify NULL, because currently the capture database must be the same as the source database. For apply rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured events, then the apply process can apply events from only one capture process at one source database. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

ADD_SCHEMA_PROPAGATION_RULES Procedure

Adds propagation rules that propagate the LCRs related to the specified schema in a source queue to a destination queue. This procedure also configures propagation using the current user, if necessary, and establishes a default propagation schedule. This procedure enables propagation of LCRs for the specified schema, subject to filtering conditions. Only one propagation job is allowed between the source queue and the destination queue.

If propagation rules are added, then the propagation job propagates DML changes, or DDL changes, or both that are related to the specified schema from the specified source queue to the specified destination queue. This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. A system-generated rule name is the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. For the overloaded `ADD_SCHEMA_PROPAGATION_RULES` procedure, the system-generated rule names for DML and DDL changes are returned.

A propagation job uses the rules created for filtering. If the propagation job does not have a rule set, then a rule set is created automatically, and the rules for propagating changes to the schema are added to the rule set. Other rules in an existing rule set for the propagation job are not affected. Additional rules can be added using the `DBMS_RULE_ADM` package.

The following is an example of a schema rule condition that may be created for propagating DML changes with a propagation job:

```
:dml.get_object_owner() = 'HR' AND :dml.is_null_tag() = 'Y'
AND :dml.get_source_database_name() = 'DBS1.NET'
```

> **Note:** The quotation marks in the preceding example are all single quotation marks.

For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate events.

Note:

- Currently, a single propagation job propagates all events that use a particular database link, even if the database link propagates events to multiple destination queues.
- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users may or may not be the same.

See Also: "CREATE_PROPAGATION Procedure" on page 47-4 for more information about the required privileges

Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
   schema_name              IN  VARCHAR2,
   streams_name             IN  VARCHAR2,
   source_queue_name        IN  VARCHAR2,
   destination_queue_name   IN  VARCHAR2,
   include_dml              IN  BOOLEAN  DEFAULT true,
   include_ddl              IN  BOOLEAN  DEFAULT false,
   include_tagged_lcr       IN  BOOLEAN  DEFAULT false,
   source_database          IN  VARCHAR2 DEFAULT NULL,
   dml_rule_name            OUT VARCHAR2,
   ddl_rule_name            OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Parameters

Table 73–4 ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

Parameter	Description
schema_name	The name of the schema. For example, hr.
streams_name	The name of the propagation job. If the specified propagation job does not exist, then it is created automatically. If NULL and a propagation job exists for the same source queue and destination queue (including database link), then this propagation job is used. If NULL and no propagation job exists for the same source queue and destination queue (including database link), then a propagation job is created automatically with a system-generated name.
source_queue_name	The name of the source queue. The current database must contain the source queue.
destination_queue_name	The name of the destination queue, including database link, for example STREAMS_QUEUE@DBS2. If the database link is omitted, then the global name of the current database is used, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then creates a rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags

Table 73–4 ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters (Cont.)

Parameter	Description
source_database	The global name of the source database. The source database is where the change originated. If NULL, then no condition regarding the source database is added to the generated rules. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically. Oracle Corporation recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

ADD_SCHEMA_RULES Procedure

Adds capture or apply rules for a schema.

If capture rules are added, then the capture process captures DML changes, or DDL changes, or both in the specified schema and enqueues these changes into the specified queue. For capture rules, you should execute this procedure at the source database. This procedure automatically invokes the PREPARE_SCHEMA_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package for the specified schema.

If apply rules are added, then the apply process receives and applies captured events that contain DML changes, or DDL changes, or both for the specified schema. For apply rules, you should execute this procedure at the destination database.

An apply process created by this procedure can apply events only at the local database and can apply only captured events. To create an apply process that applies events at a remote database or an apply process that applies user-enqueued events, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

Changes applied by an apply process created by this procedure generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

You have the option of creating an apply process using the DBMS_APPLY_ADM.CREATE_APPLY procedure and specifying nondefault values for the apply_captured, apply_database_link, and apply_tag parameters when you run that procedure. Then you can use this ADD_SCHEMA_RULES procedure to add rules to the rule set used by the apply process.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. A system-generated rule name is the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated.

The following is an example of a schema rule condition that may be created for filtering DML statements:

```
:dml.get_object_owner() = 'HR' AND :dml.is_null_tag() = 'Y'
```

Note: The quotation marks in the preceding example are all single quotation marks.

For the overloaded ADD_SCHEMA_RULES procedure, the system-generated rule names for DML and DDL changes are returned.

A capture process or an apply process uses the rules created for filtering. If the process does not have a rule set, then a rule set is created automatically, and the rules for the schema are added to the rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using the DBMS_RULE_ADM package.

If this procedure creates a capture process or an apply process, then the user who runs this procedure is the user who captures or applies changes. The specified user must have the necessary privileges to perform these actions.

See Also:

- Chapter 64, "DBMS_RULE_ADM"
- "CREATE_CAPTURE Procedure" on page 8-6 for information about the privileges required to capture changes
- "CREATE_APPLY Procedure" on page 4-9 for information about the privileges required to apply changes (refer to the apply_user parameter)

Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name              IN  VARCHAR2,
   streams_type             IN  VARCHAR2,
   streams_name             IN  VARCHAR2 DEFAULT NULL,
   queue_name               IN  VARCHAR2 DEFAULT 'streams_queue',
   include_dml              IN  BOOLEAN  DEFAULT true,
   include_ddl              IN  BOOLEAN  DEFAULT false,
   include_tagged_lcr       IN  BOOLEAN  DEFAULT false,
   source_database          IN  VARCHAR2 DEFAULT NULL,
   dml_rule_name            OUT VARCHAR2,
   ddl_rule_name            OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

Parameters

Table 73–5 ADD_SCHEMA_RULES Procedure Parameters **(Page 1 of 2)**

Parameter	Description
`schema_name`	The name of the schema. For example, `hr`. You can specify a schema that does not yet exist, because Streams does not validate the existence of the schema.
`streams_type`	The type of process, either `capture` or `apply`
`streams_name`	The name of the process. If the specified process does not exist, then it is created automatically. If `NULL` and one relevant capture or apply process for the queue exists, then the relevant process is used. If no relevant capture or apply process exists for the queue, then a capture process or an apply process is created automatically with a system-generated name. If `NULL` and multiple processes of the specified `streams_type` for the queue exist, then an error is raised.
`queue_name`	The name of the local queue. For capture rules, the queue into which the changes will be enqueued. For apply rules, the queue from which changes will be dequeued.

Table 73–5 ADD_SCHEMA_RULES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
include_dml	If TRUE, then creates a rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then a redo entry is always considered for capture and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then a redo entry is considered for capture and an LCR is considered for apply only when the redo entry or the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags
source_database	The global name of the source database. If NULL, then no condition regarding the source database is added to the generated rules. For capture rules, you can specify NULL, because currently the capture database must be the same as the source database. For apply rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured events, then the apply process can apply events from only one capture process at one source database. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

ADD_SUBSET_RULES Procedure

Adds apply rules for a subset of the rows in a table.

The apply process receives and applies captured events that contain DML changes for the specified subset of rows in the table. You should execute this procedure at the destination database.

Running this procedure generates three rules for the specified apply process: one for `INSERT` statements, one for `UPDATE` statements, and one for `DELETE` statements. For `INSERT` and `DELETE` statements, only row LCRs that satisfy the condition specified for the `dml_condition` parameter are applied. For `UPDATE` statements, the following variations are possible:

- If both the new and old values in a row LCR satisfy the specified `dml_condition`, then the LCR is applied without any changes.
- If neither the new or old values in a row LCR satisfy the specified `dml_condition`, then the row LCR is not applied.
- If the old values for a row LCR satisfy the specified `dml_condition`, but the new values do not, then the row LCR is converted into a delete.
- If the new values for a row LCR satisfy the specified `dml_condition`, but the old values do not, then the row LCR is converted to an insert.

The following is an example of a rule condition that may be created for filtering LCRs containing an update operation when the `dml_condition` is `region_id = 2` and the `table_name` is `hr.regions`:

```
:dml.get_object_owner() = 'HR' AND :dml.get_object_name() = 'REGIONS' AND
:dml.is_null_tag() = 'Y' AND :dml.get_command_type() = 'UPDATE' AND
(:dml.get_value('NEW','"REGION_ID"') IS NOT NULL) AND
(:dml.get_value('OLD','"REGION_ID"') IS NOT NULL) AND
(:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2) AND
(:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

Note: The quotation marks in the preceding example are all single quotation marks.

An apply process uses the generated rules for filtering LCRs. If the apply process does not have a rule set, then one is created automatically, and the rules for the table are added to the rule set. Other rules in an existing rule set for the apply process are not affected. Additional rules can be added using the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. The system-generated rule name is the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD_SUBSET_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

An apply process created by this procedure can apply events only at the local database and can apply only captured events. To create an apply process that applies events at a remote database or an apply process that applies user-enqueued events, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

Changes applied by an apply process created by this procedure generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

You have the option of creating an apply process using the DBMS_APPLY_ADM.CREATE_APPLY procedure and specifying nondefault values for the apply_captured, apply_database_link, and apply_tag parameters when you run that procedure. Then you can use this ADD_SUBSET_RULES procedure to add rules to the rule set used by the apply process.

When you create a subset rule for a table, you should create an unconditional supplemental log group at the source database with all the columns in the table. Supplemental logging is required if an update must be converted to an insert. The apply process must have all the column values to be able to perform the insert correctly.

If this procedure creates an apply process, then the user who runs this procedure is the user who applies changes. The specified user must have the necessary privileges to apply events.

See Also:

- Chapter 64, "DBMS_RULE_ADM"
- "CREATE_CAPTURE Procedure" on page 8-6 for information about the privileges required to capture changes
- "CREATE_APPLY Procedure" on page 4-9 for information about the privileges required to apply changes (refer to the `apply_user` parameter)

Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
   table_name                IN   VARCHAR2,
   dml_condition             IN   VARCHAR2,
   streams_type              IN   VARCHAR2 DEFAULT 'apply',
   streams_name              IN   VARCHAR2 DEFAULT NULL,
   queue_name                IN   VARCHAR2 DEFAULT 'streams_queue',
   include_tagged_lcr        IN   BOOLEAN  DEFAULT false,
   source_database           IN   VARCHAR2 DEFAULT NULL,
   insert_rule_name          OUT  VARCHAR2,
   update_rule_name          OUT  VARCHAR2,
   delete_rule_name          OUT  VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains three `OUT` parameters, and the other does not.

Parameters

Table 73–6 ADD_SUBSET_RULES Procedure Parameters **(Page 1 of 2)**

Parameter	Description
table_name	The name of the table specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default. You can specify a table that does not yet exist, because Streams does not validate the existence of the table.
dml_condition	The subset condition. You specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: `' salary > 4000 and job_id = ''SA_MAN'' '` **Note:** The quotation marks in the preceding example are all single quotation marks.
streams_type	The type of process. Currently, the only valid type is apply.
streams_name	The name of the apply process. If the specified apply process does not exist, then it is created automatically. If NULL, then the apply process for the queue is used. If no apply process exists for the queue, then one is created automatically with a system-generated name. If multiple apply processes exist, then an error is raised.
queue_name	The name of the local queue from which changes will be dequeued.
include_tagged_lcr	If TRUE, then an LCR is always considered for apply, regardless of whether the LCR has a non-NULL tag. If FALSE, then an LCR is considered for apply only when the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags

Table 73–6 ADD_SUBSET_RULES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
source_database	The global name of the source database. If NULL, then no condition regarding the source database is added to the generated rules. Specify the source database for the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured events, then the apply process can apply events from only one capture process at one source database. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically.
insert_rule_name	Contains the system-generated INSERT rule name. This rule handles insert LCRs and update LCRs that must be converted into insert LCRs.
update_rule_name	Contains the system-generated UPDATE rule name. This rule handles update LCRs that remain update LCRs.
delete_rule_name	Contains the system-generated DELETE rule name. This rule handles delete LCRs and update LCRs that must be converted into delete LCRs

ADD_TABLE_PROPAGATION_RULES Procedure

Adds propagation rules that propagate the LCRs related to the specified table in a source queue to a destination queue. This procedure also configures propagation using the current user, if necessary, and establishes a default propagation schedule. This procedure enables propagation of LCRs for the specified table, subject to filtering conditions. Only one propagation job is allowed between the source queue and the destination queue.

If propagation rules are added, then the propagation job propagates DML changes, or DDL changes, or both related to the specified table from the specified source queue to the specified destination queue. This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. A system-generated rule name is the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. For the overloaded `ADD_TABLE_PROPAGATION_RULES` procedure, the system-generated rule names for For the overloaded `ADD_SCHEMA_RULES` procedure, the system-generated rule names for DML and DDL changes are returned.

A propagation job uses the rules created for filtering. If the propagation job does not have a rule set, then a rule set is created automatically, and the rules for propagating changes to the table are added to the rule set. Other rules in an existing rule set for the propagation job are not affected. Additional rules can be added using the `DBMS_RULE_ADM` package.

The following is an example of a table rule condition that may be created for propagating DML changes with a propagation job:

```
:dml.get_object_owner() = 'HR' AND :dml.get_object_name() = 'LOCATIONS'
AND :dml.is_null_tag() = 'Y' AND :dml.get_source_database_name() = 'DBS1.NET'
```

Note: The quotation marks in the preceding example are all single quotation marks.

For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate events.

Note:

- Currently, a single propagation job propagates all events that use a particular database link, even if the database link propagates events to multiple destination queues.
- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users may or may not be the same.

See Also: "CREATE_PROPAGATION Procedure" on page 47-4 for more information about the required privileges

Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
   table_name                IN  VARCHAR2,
   streams_name              IN  VARCHAR2,
   source_queue_name         IN  VARCHAR2,
   destination_queue_name    IN  VARCHAR2,
   include_dml               IN  BOOLEAN  DEFAULT true,
   include_ddl               IN  BOOLEAN  DEFAULT false,
   include_tagged_lcr        IN  BOOLEAN  DEFAULT false,
   source_database           IN  VARCHAR2 DEFAULT NULL,
   dml_rule_name             OUT VARCHAR2,
   ddl_rule_name             OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

Parameters

Table 73–7 ADD_TABLE_PROPAGATION_RULES Procedure Parameters

Parameter	Description
table_name	The name of the table specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
streams_name	The name of the propagation job. If the specified propagation job does not exist, then it is created automatically. If NULL and a propagation job exists for the same source queue and destination queue (including database link), then this propagation job is used. If NULL and no propagation job exists for the same source queue and destination queue (including database link), then a propagation job is created automatically with a system-generated name.
source_queue_name	The name of the source queue. The current database must contain the source queue.
destination_queue_name	The name of the destination queue, including database link, for example STREAMS_QUEUE@DBS2. If the database link is omitted, then the global name of the current database is used, and the source queue and destination queue must be in the same database. **Note:** Connection qualifiers are not allowed.
include_dml	If TRUE, then creates a rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags

Table 73–7 ADD_TABLE_PROPAGATION_RULES Procedure Parameters (Cont.)

Parameter	Description
source_database	The global name of the source database. The source database is where the change originated. If NULL, then no condition regarding the source database is added to the generated rules. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically. Oracle Corporation recommends that you specify a source database for propagation rules.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

ADD_TABLE_RULES Procedure

Adds capture or apply rules for a table.

If capture rules are added, then the capture process captures DML changes, or DDL changes, or both in the specified table and enqueues these changes into the specified queue. For capture rules, you should execute this procedure at the source database. This procedure automatically invokes the PREPARE_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package for the specified table.

If apply rules are added, then the apply process receives and applies captured events that contain DML changes, or DDL changes, or both for the specified table. For apply rules, you should execute this procedure at the destination database.

An apply process created by this procedure can apply events only at the local database and can apply only captured events. To create an apply process that applies events at a remote database or an apply process that applies user-enqueued events, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

Changes applied by an apply process created by this procedure generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

You have the option of creating an apply process using the DBMS_APPLY_ADM.CREATE_APPLY procedure and specifying nondefault values for the apply_captured, apply_database_link, and apply_tag parameters when you run that procedure. Then you can use this ADD_TABLE_RULES procedure to add rules to the rule set used by the apply process.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. A system-generated rule name is the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated.

For example, the following is an example of a rule condition that may be created for filtering DML statements:

```
:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'EMPLOYEES'
AND :dml.is_null_tag() = 'Y' AND :dml.get_source_database_name() = 'DBS1.NET'
```

Note: The quotation marks in the preceding example are all single quotation marks.

For the overloaded ADD_TABLE_RULES procedure, the system-generated rule names for DML and DDL changes are returned.

A capture process or an apply process uses the rules created for filtering. If the process does not have a rule set, then a rule set is created automatically, and the rules for the table are added to the rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using the DBMS_RULE_ADM package.

If this procedure creates a capture process or an apply process, then the user who runs this procedure is the user who captures or applies changes. The specified user must have the necessary privileges to perform these actions.

See Also:

- Chapter 64, "DBMS_RULE_ADM"
- "CREATE_CAPTURE Procedure" on page 8-6 for information about the privileges required to capture changes
- "CREATE_APPLY Procedure" on page 4-9 for information about the privileges required to apply changes (refer to the apply_user parameter)

Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
   table_name                IN  VARCHAR2,
   streams_type              IN  VARCHAR2,
   streams_name              IN  VARCHAR2 DEFAULT NULL,
   queue_name                IN  VARCHAR2 DEFAULT 'streams_queue',
   include_dml               IN  BOOLEAN  DEFAULT true,
   include_ddl               IN  BOOLEAN  DEFAULT false,
   include_tagged_lcr        IN  BOOLEAN  DEFAULT false,
   source_database           IN  VARCHAR2 DEFAULT NULL,
   dml_rule_name             OUT VARCHAR2,
   ddl_rule_name             OUT VARCHAR2);
```

Note: This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

Parameters

Table 73–8 ADD_TABLE_RULES Procedure Parameters (Page 1 of 2)

Parameter	Description
table_name	The name of the table specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default. You can specify a table that does not yet exist, because Streams does not validate the existence of the table.
streams_type	The type of process, either capture or apply
streams_name	The name of the process. If the specified process does not exist, then it is created automatically. If NULL and one relevant capture or apply process for the queue exists, then the relevant capture or apply process is used. If no relevant process exists for the queue, then a capture process or an apply process is created automatically with a system-generated name. If NULL and multiple processes of the specified streams_type for the queue exist, then an error is raised.
queue_name	The name of the local queue. For capture rules, the queue into which the changes will be enqueued. For apply rules, the queue from which changes will be dequeued.
include_dml	If TRUE, then creates a DML rule for DML changes. If FALSE, then does not create a DML rule. NULL is not permitted.
include_ddl	If TRUE, then creates a DDL rule for DDL changes. If FALSE, then does not create a DDL rule. NULL is not permitted.
include_tagged_lcr	If TRUE, then a redo entry is always considered for capture and an LCR is always considered for apply, regardless of whether redo entry or LCR has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then a redo entry is considered for capture and an LCR is considered for apply only when the redo entry or the LCR contains a NULL tag. A setting of false is often specified in update-anywhere configurations to avoid sending a change back to its source database. **See Also:** *Oracle9i Streams* for more information about tags

Table 73–8 ADD_TABLE_RULES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
source_database	The global name of the source database. If NULL, then no condition regarding the source database is added to the generated rules. For capture rules, you can specify NULL, because currently the capture database must be the same as the source database. For apply rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured events, then the apply process can apply events from only one capture process at one source database. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically.
dml_rule_name	If include_dml is TRUE, then contains the DML rule name. If include_dml is FALSE, then contains a NULL.
ddl_rule_name	If include_ddl is TRUE, then contains the DDL rule name. If include_ddl is FALSE, then contains a NULL.

PURGE_SOURCE_CATALOG Procedure

Removes all Streams data dictionary information at the local database for the specified object. You can use this procedure to remove Streams metadata that is not needed currently and will not be needed in the future.

The global name of the source database containing the object must be specified for the `source_database` parameter. If the current database is not the source database for the object, then data dictionary information about the object is removed at the current database, not the source database.

For example, suppose changes to the `hr.employees` table at the `dbs1.net` source database are being applied to the `hr.employees` table at the `dbs2.net` destination database. Also, suppose `hr.employees` at `dbs2.net` is not a source at all. In this case, specifying `dbs2.net` as the `source_database` for this table results in an error. However, specifying `dbs1.net` as the `source_database` for this table while running the `PURGE_SOURCE_CATALOG` procedure at the `dbs2.net` database removes data dictionary information about the table at `dbs2.net`.

Do not run this procedure at a database if either of the following conditions are true:

- LCRs captured by the capture process for the object are or may be applied locally without reinstantiating the object.
- LCRs captured by the capture process for the object are or may be forwarded by the database without reinstantiating the object.

> **Note:** These conditions do not apply to LCRs that were not created by the capture process. That is, these conditions do not apply to user-created LCRs.

Syntax

```
DBMS_STREAMS_ADM.PURGE_SOURCE_CATALOG(
   source_database        IN VARCHAR2,
   source_object_name     IN VARCHAR2,
   source_object_type     IN VARCHAR2);
```

Parameters

Table 73–9 PURGE_SOURCE_CATALOG Procedure Parameters

Parameter	Description
source_database	The global name of the source database containing the object. If you do not include the domain name, then it is appended to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then DBS1.NET is specified automatically.
source_object_name	The name of the object specified as [*schema_name*.]*object_name*. For example, hr.employees. If the schema is not specified, then the current user is the default.
source_object_type	Type of the object. Currently, TABLE is the only possible object type.

REMOVE_RULE Procedure

Removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, or propagation job.

> **Note:** If a rule was automatically created by the system, then you should use this procedure to remove the rule instead of the DBMS_RULE_ADM.REMOVE_RULE procedure. If you use the DBMS_RULE_ADM.REMOVE_RULE procedure, then some metadata about the rule may remain.

Syntax

```
DBMS_STREAMS_ADM.REMOVE_RULE(
   rule_name             IN VARCHAR2,
   streams_type          IN VARCHAR2,
   streams_name          IN VARCHAR2,
   drop_unused_rule      IN BOOLEAN DEFAULT true);
```

Parameters

Table 73–10 REMOVE_RULE Procedure Parameters

Parameter	Description
rule_name	The name of the rule to remove. If NULL, then removes all rules for the specified capture process, apply process, or propagation job rule set.
streams_type	The type of Streams rule, either capture, apply, or propagate
streams_name	The name of the capture process, apply process, or propagation job
drop_unused_rule	If false, then the rule is not dropped from the database. If true and the rule is not in any rule set, then the rule is dropped from the database. If true and the rule exists in any rule set, then the rule is not dropped from the database.

SET_UP_QUEUE Procedure

Creates a queue table and a Streams queue for use with the capture, propagate, and apply functionality of Streams.

Set up includes the following actions:

- If the specified queue table does not exist, then this procedure runs the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create the queue table with the specified storage clause.
- If the specified queue name does not exist, then this procedure runs the CREATE_QUEUE procedure in the DBMS_AQADM package to create the queue.
- This procedure starts the queue.
- If a queue user is specified, then this procedure configures this user as a secure queue user of the queue and grants ENQUEUE and DEQUEUE privileges on the queue to the specified queue user.

  To configure the queue user as a secure queue user, this procedure creates an Advanced Queuing agent with the same name as the user name, if one does not already exist. If an agent with this name already exists and is associated with the queue user only, then it is used. SET_UP_QUEUE then runs the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the agent and the user.

This procedure creates a SYS.AnyData queue that is both a secure queue and a transactional queue.

Note:

- To enqueue events into and dequeue events from a queue, a queue user must have EXECUTE privilege on the DBMS_AQ package. The SET_UP_QUEUE procedure does not grant this privilege.
- If the agent that SET_UP_QUEUE tries to create already exists and is associated with a user other than the user specified by queue_user, then an error is raised. In this case, rename or remove the existing agent, and retry SET_UP_QUEUE.

See Also: The GRANT_QUEUE_PRIVILEGE procedure in the chapter describing the DBMS_AQADM package for more information about these privileges

Syntax

```
DBMS_STREAMS_ADM.SET_UP_QUEUE(
   queue_table          IN VARCHAR2 DEFAULT 'streams_queue_table',
   storage_clause       IN VARCHAR2 DEFAULT NULL,
   queue_name           IN VARCHAR2 DEFAULT 'streams_queue',
   queue_user           IN VARCHAR2 DEFAULT NULL,
   comment              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 73–11 SET_UP_QUEUE Procedure Parameters

Parameter	Description
queue_table	The name of the queue table specified as [*schema_name.*]*queue_table_name*. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If the queue table owner is not specified, then the user who runs this procedure is automatically specified as the queue table owner.
storage_clause	The storage clause for queue table The storage parameter is included in the CREATE TABLE statement when the queue table is created. You can specify any valid table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace of the user who runs this procedure. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. If NULL, then Oracle uses the storage characteristics of the tablespace in which the queue table is created. **See Also:** *Oracle9i SQL Reference* for more information about storage clauses
queue_name	The name of the queue that will function as the Streams queue, specified as [*schema_name.*]*queue_name*. For example, strmadmin.streams_queue. If the schema is not specified, then the current user is the default. If the queue owner is not specified, then it defaults to the queue table owner. The owner of the queue table must also be the owner of the queue. The queue owner automatically has privileges to perform all queue operations on the queue.
queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue. This user is also configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then no privileges are granted. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
comment	The comment for the queue

74

DBMS_TRACE

Oracle8*i* PL/SQL provides an API for tracing the execution of PL/SQL programs on the server. You can use the trace API, implemented on the server as the DBMS_TRACE package, to trace PL/SQL functions, procedures, and exceptions.

DBMS_TRACE provides subprograms to start and stop PL/SQL tracing in a session. Oracle collects the trace data as the program executes and writes it to database tables.

A typical session involves:

- Starting PL/SQL tracing in session (DBMS_TRACE.SET_PLSQL_TRACE).
- Running an application to be traced.
- Stopping PL/SQL tracing in session (DBMS_TRACE.CLEAR_PLSQL_TRACE).

This chapter discusses the following topics:

Requirements, Restrictions, and Constants for DBMS_TRACE

Requirements

This package must be created under SYS.

Restrictions

You cannot use PL/SQL tracing in a shared server environment.

Constants

DBMS_TRACE uses these constants:

```
trace_all_calls          constant INTEGER := 1;
trace_enabled_calls      constant INTEGER := 2;
trace_all_exceptions     constant INTEGER := 4;
trace_enabled_exceptions constant INTEGER := 8;
trace_all_sql            constant INTEGER := 32;
trace_enabled_sql        constant INTEGER := 64;
trace_all_lines          constant INTEGER := 128;
trace_enabled_lines      constant INTEGER := 256;
trace_stop               constant INTEGER := 16384;
trace_pause              constant INTEGER := 4096;
trace_resume             constant INTEGER := 8192;
trace_limit              constant INTEGER := 16;
trace_major_version      constant BINARY_INTEGER := 1;
trace_minor_version      constant BINARY_INTEGER := 0;
```

Oracle recommends using the symbolic form for all these constants.

Using DBMS_TRACE

Controlling Data Volume

Profiling large applications may produce a large volume of data. You can control the volume of data collected by enabling specific program units for trace data collection.

You can enable a program unit by compiling it debug. This can be done in one of two ways:

```
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be
```

```
generated */
```

or:

```
/* recompile specific library unit with debug option */
alter [PROCEDURE | FUNCTION | PACKAGE BODY]  <libunit-name> compile debug;
```

Note: You cannot use the second method for anonymous blocks.

You can limit the amount of storage used in the database by retaining only the most recent 8,192 records (approximately) by including `TRACE_LIMIT` in the `TRACE_LEVEL` parameter of the `SET_PLSQL_TRACE` procedure.

Creating Database Tables to Collect DBMS_TRACE Output

You must create database tables into which the DBMS_TRACE package writes output. Otherwise, the data is not collected. To create these tables, run the script `TRACETAB.SQL`. The tables this script creates are owned by SYS.

Collecting Trace Data

The PL/SQL features you can trace are described in the script `DBMSPBT.SQL`. Some of the key tracing features are:

- Tracing Calls
- Tracing Exceptions
- Tracing SQL
- Tracing Lines

Additional features of DBMS_TRACE also allow pausing and resuming trace, and limiting the output.

Tracing Calls

Two levels of call tracing are available:

- Level 1: Trace all calls. This corresponds to the constant `trace_all_calls`.
- Level 2: Trace calls to enabled program units only. This corresponds to the constant `trace_enabled_calls`.

Enabling cannot be detected for remote procedure calls (RPCs); hence, RPCs are only traced with level 1.

Tracing Exceptions

Two levels of exception tracing are available:

- Level 1: Trace all exceptions. This corresponds to `trace_all_exceptions`.
- Level 2: Trace exceptions raised in enabled program units only. This corresponds to `trace_enabled_exceptions`.

Tracing SQL

Two levels of SQL tracing are available:

- Level 1: Trace all SQL. This corresponds to the constant `trace_all_sql`.
- Level 2: Trace SQL in enabled program units only. This corresponds to the constant `trace_enabled_sql`.

Tracing Lines

Two levels of line tracing are available:

- Level 1: Trace all lines. This corresponds to the constant `trace_all_lines`.
- Level 2: Trace lines in enabled program units only. This corresponds to the constant `trace_enabled_lines`.

When tracing lines, Oracle adds a record to the database each time the line number changes. This includes line number changes due to procedure calls and returns.

Note: For both all types of tracing, level 1 overrides level 2. For example, if both level 1 and level 2 are enabled, then level 1 takes precedence.

Collected Data

If tracing is requested only for enabled program units, and if the current program unit is not enabled, then no trace data is written.

When tracing calls, both the call and return are traced. The check for whether tracing is "enabled" passes if either the called routine or the calling routine is "enabled".

Call tracing will always output the program unit type, program unit name, and line number for both the caller and the callee. It will output the caller's stack depth. If the caller's unit is enabled, the calling procedure name will also be output. If the callee's unit is enabled, the called procedure name will be output

Exception tracing writes out the line number. Raising the exception shows information on whether the exception is user-defined or pre-defined. It also shows the exception number in the case of pre-defined exceptions. Both the place where the exceptions are raised and their handler is traced. The check for tracing being "enabled" is done independently for the place where the exception is raised and the place where the exception is handled.

All calls to DBMS_TRACE.SET_PLSQL_TRACE and DBMS_TRACE.CLEAR_PLSQL_TRACE place a special trace record in the database. Therefore, it is always possible to determine when trace settings were changed.

Trace Control

As well as determining which items are collected, you can pause and resume the trace process. No information is gathered between the time that tracing is paused and the time that it is resumed. The constants TRACE_PAUSE and TRACE_RESUME are used to accomplish this. Trace records are generated to indicate that the trace was paused / resumed.

It is also possible to retain only the last 8,192 trace events of a run by using the constant TRACE_LIMIT. This allows tracing to be turned on without filling up the database. When tracing stops, the last 8,192 records are saved. The limit is approximate, since it is not checked on every trace record. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated.

Summary of DBMS_TRACE Subprograms

Table 74–1 DBMS_TRACE Subprograms

Subprogram	Description
SET_PLSQL_TRACE Procedure on page 74-6	Starts tracing in the current session.
CLEAR_PLSQL_TRACE Procedure on page 74-6	Stops trace data dumping in session.
PLSQL_TRACE_VERSION Procedure on page 74-6	Gets the version number of the trace package.

SET_PLSQL_TRACE Procedure

This procedure enables PL/SQL trace data collection.

Syntax

```
DBMS_TRACE.SET_PLSQL_TRACE (
   trace_level INTEGER);
```

Parameters

Table 74–2 SET_PLSQL_TRACE Procedure Parameters

Parameter	Description
trace_level	You must supply one or more of the constants as listed on page 74-2. By summing the constants, you can enable tracing of multiple PL/SQL language features simultaneously. The control constants "trace_pause", "trace_resume" and "trace_stop" should not be used in combination with other constants Also see "Collecting Trace Data" on page 74-3 for more information.

CLEAR_PLSQL_TRACE Procedure

This procedure disables trace data collection.

Syntax

```
DBMS_TRACE.CLEAR_PLSQL_TRACE;
```

PLSQL_TRACE_VERSION Procedure

This procedure gets the version number of the trace package. It returns the major and minor version number of the DBMS_TRACE package.

Syntax

```
DBMS_TRACE.PLSQL_TRACE_VERSION (
   major OUT BINARY_INTEGER,
   minor OUT BINARY_INTEGER);
```

Parameters

Table 74–3 PLSQL_TRACE_VERSION Procedure Parameters

Parameter	Description
major	Major version number of DBMS_TRACE.
minor	Minor version number of DBMS_TRACE.

75

DBMS_TRANSACTION

This package provides access to SQL transaction statements from stored procedures.

See Also: *Oracle9i SQL Reference*

This chapter discusses the following topics:

- Requirements
- Summary of DBMS_TRANSACTION Subprograms

Requirements

This package runs with the privileges of calling user, rather than the package owner SYS.

Summary of DBMS_TRANSACTION Subprograms

Table 75–1 DBMS_TRANSACTION Subprograms

Subprogram
READ_ONLY Procedure on page 75-2
READ_WRITE Procedure on page 75-3
ADVISE_ROLLBACK Procedure on page 75-3
ADVISE_NOTHING Procedure on page 75-3
ADVISE_COMMIT Procedure on page 75-3
USE_ROLLBACK_SEGMENT Procedure on page 75-4
COMMIT_COMMENT Procedure on page 75-4
COMMIT_FORCE Procedure on page 75-5
COMMIT Procedure on page 75-5
SAVEPOINT Procedure on page 75-5
ROLLBACK Procedure on page 75-6
ROLLBACK_SAVEPOINT Procedure on page 75-6
ROLLBACK_FORCE Procedure on page 75-7
BEGIN_DISCRETE_TRANSACTION Procedure on page 75-7
PURGE_MIXED Procedure on page 75-8
PURGE_LOST_DB_ENTRY Procedure on page 75-9
LOCAL_TRANSACTION_ID Function on page 75-11
STEP_ID Function on page 75-11

READ_ONLY Procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION READ ONLY
```

Syntax

```
DBMS_TRANSACTION.READ_ONLY;
```

READ_WRITE Procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION READ WRITE
```

Syntax

```
DBMS_TRANSACTION.READ_WRITE;
```

ADVISE_ROLLBACK Procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE ROLLBACK
```

Syntax

```
DBMS_TRANSACTION.ADVISE_ROLLBACK;
```

ADVISE_NOTHING Procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE NOTHING
```

Syntax

```
DBMS_TRANSACTION.ADVISE_NOTHING;
```

ADVISE_COMMIT Procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE COMMIT
```

Syntax

```
DBMS_TRANSACTION.ADVISE_COMMIT;
```

USE_ROLLBACK_SEGMENT Procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>
```

Syntax

```
DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT (
   rb_name VARCHAR2);
```

Parameters

Table 75–2 USE_ROLLBACK_SEGMENT Procedure Parameters

Parameter	Description
rb_name	Name of rollback segment to use.

COMMIT_COMMENT Procedure

This procedure is equivalent to following SQL statement:

```
COMMIT COMMENT <text>
```

Syntax

```
DBMS_TRANSACTION.COMMIT_COMMENT (
   cmnt VARCHAR2);
```

Parameters

Table 75–3 COMMIT_COMMENT Procedure Parameters

Parameter	Description
cmnt	Comment to associate with this commit.

COMMIT_FORCE Procedure

This procedure is equivalent to following SQL statement:

```
COMMIT FORCE <text>, <number>"
```

Syntax

```
DBMS_TRANSACTION.COMMIT_FORCE (
   xid VARCHAR2,
   scn VARCHAR2 DEFAULT NULL);
```

Parameters

Table 75–4 COMMIT_FORCE Procedure Parameters

Parameter	Description
xid	Local or global transaction ID.
scn	System change number.

COMMIT Procedure

This procedure is equivalent to following SQL statement:

```
COMMIT
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.COMMIT;
```

SAVEPOINT Procedure

This procedure is equivalent to following SQL statement:

```
SAVEPOINT <savepoint_name>
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 75–5 SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

ROLLBACK Procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK;
```

ROLLBACK_SAVEPOINT Procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK TO SAVEPOINT <savepoint_name>
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 75–6 ROLLBACK_SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

ROLLBACK_FORCE Procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK FORCE <text>
```

Syntax

```
DBMS_TRANSACTION.ROLLBACK_FORCE (
   xid VARCHAR2);
```

Parameters

Table 75–7 ROLLBACK_FORCE Procedure Parameters

Parameter	Description
xid	Local or global transaction ID.

BEGIN_DISCRETE_TRANSACTION Procedure

This procedure sets "discrete transaction mode" for this transaction.

Syntax

```
DBMS_TRANSACTION.BEGIN_DISCRETE_TRANSACTION;
```

Exceptions

Table 75–8 BEGIN_DISCRETE_TRANSACTION Procedure Exceptions

Exception	Description
ORA-08175	A transaction attempted an operation which cannot be performed as a discrete transaction. If this exception is encountered, then rollback and retry the transaction
ORA-08176	A transaction encountered data changed by an operation that does not generate rollback data: create index, direct load or discrete transaction. If this exception is encountered, then retry the operation that received the exception.

Example

```
DISCRETE_TRANSACTION_FAILED exception;
    pragma exception_init(DISCRETE_TRANSACTION_FAILED, -8175);
CONSISTENT_READ_FAILURE exception;
    pragma exception_init(CONSISTENT_READ_FAILURE, -8176);
```

PURGE_MIXED Procedure

When in-doubt transactions are forced to commit or rollback (instead of letting automatic recovery resolve their outcomes), there is a possibility that a transaction can have a mixed outcome: Some sites commit, and others rollback. Such inconsistency cannot be resolved automatically by Oracle; however, Oracle flags entries in DBA_2PC_PENDING by setting the MIXED column to a value of 'yes'.

Oracle never automatically deletes information about a mixed outcome transaction. When the application or DBA is certain that all inconsistencies that might have arisen as a result of the mixed transaction have been resolved, this procedure can be used to delete the information about a given mixed outcome transaction.

Syntax

```
DBMS_TRANSACTION.PURGE_MIXED (
   xid VARCHAR2);
```

Parameters

Table 75–9 PURGE_MIXED Procedure Parameters

Parameter	Description
xid	Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.

PURGE_LOST_DB_ENTRY Procedure

When a failure occurs during commit processing, automatic recovery consistently resolves the results at all sites involved in the transaction. However, if the remote database is destroyed or re-created before recovery completes, then the entries used to control recovery in DBA_2PC_PENDING and associated tables are never removed, and recovery will periodically retry. Procedure PURGE_LOST_DB_ENTRY enables removal of such transactions from the local site.

Syntax

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY (
   xid VARCHAR2);
```

WARNING: PURGE_LOST_DB_ENTRY should *only* be used when the other database is lost or has been re-created. Any other use may leave the other database in an unrecoverable or inconsistent state.

Before automatic recovery runs, the transaction may show up in DBA_2PC_ PENDING as state "collecting", "committed", or "prepared". If the DBA has forced an in-doubt transaction to have a particular result by using "commit force" or "rollback force", then states "forced commit" or "forced rollback" may also appear. Automatic recovery normally deletes entries in any of these states. The only exception is when recovery finds a forced transaction which is in a state inconsistent with other sites in the transaction; in this case, the entry is left in the table and the MIXED column has the value 'yes'.

However, under certain conditions, it may not be possible for automatic recovery to run. For example, a remote database may have been permanently lost. Even if it is re-created, it gets a new database ID, so that recovery cannot identify it (a possible symptom is ORA-02062). In this case, the DBA may use the procedure PURGE_

LOST_DB_ENTRY to clean up the entries in any state other than "prepared". The DBA does not need to be in any particular hurry to resolve these entries, because they are not holding any database resources.

The following table indicates what the various states indicate about the transaction and what the DBA actions should be:

Table 75–10 PURGE_LOST_DB_ENTRY Procedure States

State of Column	State of Global Transaction	State of Local Transaction	Normal DBA Action	Alternative DBA Action
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Committed	Committed	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Prepared	Unknown	Prepared	None	FORCE COMMIT or ROLLBACK
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced commit (mixed)	Mixed	Committed	(See Note 2)	
Forced rollback (mixed)	Mixed	Rolled back	(See Note 2)	

NOTE 1: Use only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples are total loss of the remote database, reconfiguration in software resulting in loss of two-phase commit capability, or loss of information from an external transaction coordinator such as a TP monitor.

NOTE 2: Examine and take any manual action to remove inconsistencies; then use the procedure PURGE_MIXED.

Parameters

Table 75–11 PURGE_LOST_DB_ENTRY Procedure Parameters

Parameter	Description
xid	Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.

LOCAL_TRANSACTION_ID Function

This function returns the local (to instance) unique identifier for current transaction. It returns null if there is no current transaction.

Syntax

```
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (
   create_transaction BOOLEAN := FALSE)
  RETURN VARCHAR2;
```

Parameters

Table 75–12 LOCAL_TRANSACTION_ID Function Parameters

Parameter	Description
create_ transaction	If true, then start a transaction if one is not currently active.

STEP_ID Function

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

Syntax

```
DBMS_TRANSACTION.STEP_ID
   RETURN NUMBER;
```

76

DBMS_TRANSFORM

The DBMS_TRANSFORM package provides an interface to the message format transformation features of Oracle Advanced Queuing.

> **See Also:** *Oracle9i Application Developer's Guide - Advanced Queuing* for more on message format transformations.

This chapter discusses the following topics:

- Summary of DBMS_TRANSFORM Subprograms

Summary of DBMS_TRANSFORM Subprograms

Table 76–1 DBMS_TRANSFORM Subprograms

Subprograms	Description
CREATE_ TRANSFORMATION Procedure on page 76-2	Creates a transformation that maps an object of the source type to an object of the destination type
MODIFY_ TRANSFORMATION Procedure on page 76-3	Modifies an existing transformation
DROP_ TRANSFORMATION Procedure on page 76-4	Drops the given transformation

CREATE_TRANSFORMATION Procedure

This procedure creates a transformation that maps an object of the source type to an object of the target type. The transformation expression can be a SQL expression or a PL/SQL function. It must return an object of the target type.

Syntax

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION (
    schema              VARCHAR2(30),
    name                VARCHAR2(30),
    from_schema         VARCHAR2(30),
    from_type           VARCHAR2(30),
    to_schema           VARCHAR2(30),
    to_type             VARCHAR2(30),
    transformation      VARCHAR2(4000));
```

Parameters

Table 76–2 CREATE_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation
name	Specifies the name of the transformation
from_schema	Specifies the schema of the source type

Table 76–2 CREATE_TRANSFORMATION Procedure Parameters

Parameter	Description
from_type	Specifies the source type
to_schema	Specifies the target type schema
to_type	Specifies the target type
transformation	Specifies the transformation expression, returning an object of the target type. If the target type is an ADT, the expression must be a function returning an object of the target type or a constructor expression for the target type. You can choose not to specify a transformation expression and instead specify transformations for attributes of the target type using MODIFY_TRANSFORMATION.

MODIFY_TRANSFORMATION Procedure

This procedure modifies (or creates) the mapping for the specified attribute of the target type. The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type. An attribute number zero must be specified for a scalar target type. If the target type is an ADT, and the attribute_number is zero, then the expression must be a PL/SQL function returning an object of the target type or a constructor expression for the target type.

Syntax

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (
    schema           VARCHAR2(30)
    name             VARCHAR2(30),
    attribute_number INTEGER,
    transformation   VARCHAR2(4000));
```

Parameters

Table 76–3 MODIFY_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation
name	Specifies the name of the transformation
attribute_number	Must be zero for a scalar target type

Table 76–3 MODIFY_TRANSFORMATION Procedure Parameters

Parameter	Description
transformation	The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type

DROP_TRANSFORMATION Procedure

This procedure drops the given transformation.

Syntax

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
    schema      VARCHAR2(30),
    name        VARCHAR2(30));
```

Parameters

Table 76–4 DROP_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation
name	Specifies the name of the transformation

77

DBMS_TTS

This package checks if the transportable set is self-contained. All violations are inserted into a temporary table that can be selected from the view `TRANSPORT_SET_VIOLATIONS`.

Only users having the `execute_catalog_role` can execute this procedure. This role is initially only assigned to user SYS.

> **See Also:** *Oracle9i Database Administrator's Guide* and *Oracle9i Database Migration*

This chapter discusses the following topics:

Exceptions

```
ts_not_found  EXCEPTION;
PRAGMA exception_init(ts_not_found, -29304);
ts_not_found_num NUMBER := -29304;

invalid_ts_list  EXCEPTION;
PRAGMA exception_init(invalid_ts_list, -29346);
invalid_ts_list_num NUMBER := -29346;

sys_or_tmp_ts     EXCEPTION;
PRAGMA exception_init(sys_or_tmp_ts, -29351);
sys_or_tmp_ts_num NUMBER := -29351;
```

Summary of DBMS_TTS Subprograms

These two procedures are designed to be called by database administrators.

Table 77–1 DBMS_TTS Subprograms

Subprogram	Description
TRANSPORT_SET_CHECK Procedure on page 77-2	Checks if a set of tablespaces (to be transported) is self-contained.
DOWNGRADE Procedure on page 77-3	Downgrades transportable tablespace related data.

TRANSPORT_SET_CHECK Procedure

This procedure checks if a set of tablespaces (to be transported) is self-contained. After calling this procedure, the user may select from a view to see a list of violations, if there are any. If the view does not return any rows, then the set of tablespaces is self-contained. For example,

```
SQLPLUS> EXECUTE TRANSPORT_SET_CHECK('foo,bar', TRUE);
SQLPLUS> SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

Syntax

```
DBMS_TTS.TRANSPORT_SET_CHECK (
   ts_list          IN VARCHAR2,
   incl_constraints IN BOOLEAN DEFAULT,
   full_closure     IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 77–2 TRANSPORT_SET_CHECK Procedure Parameters

Parameter	Description
ts_list	List of tablespace, separated by comma.
incl_constraints	TRUE if you want to count in referential integrity constraints when examining if the set of tablespaces is self-contained. (The incl_constraints parameter is a default so that TRANSPORT_SET_CHECK will work if it is called with only the ts_list argument.)
full_closure	Indicates whether a full or partial dependency check is required. If TRUE, treats all IN and OUT pointers (dependencies) and captures them as violations if they are not self-contained in the transportable set. The parameter should be set to TRUE for TSPITR or if a strict version of transportable is desired. By default the parameter is set to false. It will only consider OUT pointers as violations.

DOWNGRADE Procedure

This procedure downgrades transportable tablespace related data.

Syntax

```
DBMS_TTS.DOWNGRADE;
```

78

DBMS_TYPES

The DBMS_TYPES package consists of constants, which represent the built-in and user-defined types. See *Oracle interMedia User's Guide and Reference* for a complete discussion of types.

This chapter discusses the following topics:

Constants for DBMS_TYPES

The following table lists the constants in the DBMS_TYPES package.

Table 78–1 DBMS_TYPES Constants

Constant	Description
TYPECODE_DATE	A DATE type
TYPECODE_NUMBER	A NUMBER type
TYPECODE_RAW	A RAW type
TYPECODE_CHAR	A CHAR type
TYPECODE_VARCHAR2	A VARCHAR2 type
TYPECODE_VARCHAR	A VARCHAR type
TYPECODE_MLSLABEL	An MLSLABEL type
TYPECODE_BLOB	A BLOB type
TYPECODE_BFILE	A BFILE type
TYPECODE_CLOB	A CLOB type
TYPECODE_CFILE	A CFILE type
TYPECODE_TIMESTAMP	A TIMESTAMP type
TYPECODE_TIMESTAMP_TZ	A TIMESTAMP_TZ type
TYPECODE_TIMESTAMP_LTZ	A TIMESTAMP_LTZ type
TYPECODE_INTERVAL_YM	A INTERVAL_YM type
TYPECODE_INTERVAL_DS	An INTERVAL_DS type
TYPECODE_REF	A REF type
TYPECODE_OBJECT	An OBJECT type
TYPECODE_VARRAY	A VARRAY collection type
TYPECODE_TABLE	A nested table collection type
TYPECODE_NAMEDCOLLECTION	
TYPECODE_OPAQUE	An OPAQUE type
SUCCESS	

Table 78–1 DBMS_TYPES Constants

Constant	Description
NO_DATA	

Exceptions

- INVALID_PARAMETERS
- INCORRECT_USAGE
- TYPE_MISMATCH

79

DBMS_UTILITY

This package provides various utility subprograms.

DBMS_UTILITY submits a job for each partition. It is the users responsibility to control the number of concurrent jobs by setting the INIT.ORA parameter JOB_QUEUE_PROCESSES correctly.There is minimal error checking for correct syntax. Any error is reported in SNP trace files.

This chapter discusses the following topics:

Requirements and Types for DBMS_UTILITY

Requirements

DBMS_UTILITY runs with the privileges of the calling user for the NAME_RESOLVE, COMPILE_SCHEMA, and ANALYZE_SCHEMA procedures. This is necessary so that the SQL works correctly.

This does not run as SYS. The privileges are checked using DBMS_DDL.

Types

```
type uncl_array IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```
Lists of "USER"."NAME"."COLUMN"@LINK should be stored here.

```
type name_array IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```
Lists of NAME should be stored here.

```
type dblink_array IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```
Lists of database links should be stored here.

```
TYPE index_table_type IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```
The order in which objects should be generated is returned here.

```
TYPE number_array IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```
The order in which objects should be generated is returned here for users.

```
  TYPE instance_record IS RECORD (
       inst_number   NUMBER,
       inst_name     VARCHAR2(60));
  TYPE instance_table IS TABLE OF instance_record INDEX BY BINARY_INTEGER;
```
The list of active instance number and instance name.

The starting index of instance_table is 1; instance_table is dense.

Summary of DBMS_UTILITY Subprograms

Table 79–1 DBMS_UTILITY Subprograms

Subprogram	Description
COMPILE_SCHEMA Procedure on page 79-4	Compiles all procedures, functions, packages, and triggers in the specified schema.
ANALYZE_SCHEMA Procedure on page 79-5	Analyzes all the tables, clusters, and indexes in a schema.

Table 79–1 DBMS_UTILITY Subprograms

Subprogram	Description
ANALYZE_DATABASE Procedure on page 79-6	Analyzes all the tables, clusters, and indexes in a database.
FORMAT_ERROR_STACK Function on page 79-7	Formats the current error stack.
FORMAT_CALL_STACK Function on page 79-7	Formats the current call stack.
IS_CLUSTER_DATABASE Function on page 79-7	Finds out if this database is running in cluster database mode.
GET_TIME Function on page 79-8	Finds out the current time in 100th's of a second.
GET_PARAMETER_VALUE Function on page 79-8	Gets the value of specified init.ora parameter.
NAME_RESOLVE Procedure on page 79-9	Resolves the given name.
NAME_TOKENIZE Procedure on page 79-11	Calls the parser to parse the given name.
COMMA_TO_TABLE Procedure on page 79-12	Converts a comma-delimited list of names into a PL/SQL table of names.
TABLE_TO_COMMA Procedure on page 79-12	Converts a PL/SQL table of names into a comma-delimited list of names.
PORT_STRING Function on page 79-13	Returns a string that uniquely identifies the version of Oracle and the operating system.
DB_VERSION Procedure on page 79-13	Returns version information for the database.
MAKE_DATA_BLOCK_ ADDRESS Function on page 79-14	Creates a data block address given a file number and a block number.
DATA_BLOCK_ADDRESS_FILE Function on page 79-15	Gets the file number part of a data block address.
DATA_BLOCK_ADDRESS_ BLOCK Function on page 79-15	Gets the block number part of a data block address.
GET_HASH_VALUE Function on page 79-16	Computes a hash value for the given string.
ANALYZE_PART_OBJECT Procedure on page 79-17	

Table 79–1 DBMS_UTILITY Subprograms

Subprogram	Description
EXEC_DDL_STATEMENT Procedure on page 79-18	Executes the DDL statement in `parse_string`.
CURRENT_INSTANCE Function on page 79-18	Returns the current connected instance number.
ACTIVE_INSTANCES Procedure on page 79-19	

COMPILE_SCHEMA Procedure

This procedure compiles all procedures, functions, packages, and triggers in the specified schema. After calling this procedure, you should select from view `ALL_OBJECTS` for items with status of `INVALID` to see if all objects were successfully compiled.

To see the errors associated with `INVALID` objects, you may use the Enterprise Manager command:

```
SHOW ERRORS <type> <schema>.<name>
```

Syntax

```
DBMS_UTILITY.COMPILE_SCHEMA (
   schema VARCHAR2);
```

Parameters

Table 79–2 COMPILE_SCHEMA Procedure Parameters

Parameter	Description
`schema`	Name of the schema.

Exceptions

Table 79–3 COMPILE_SCHEMA Procedure Exceptions

Exception	Description
`ORA-20000`	Insufficient privileges for some object in this schema.

ANALYZE_SCHEMA Procedure

This procedure runs the ANALYZE command on all the tables, clusters, and indexes in a schema. Use this procedure to collect nonoptimizer statistics. For optimizer statistics, use the DBMS_STATS.GATHER_SCHEMA_STATS procedure.

Syntax

```
DBMS_UTILITY.ANALYZE_SCHEMA (
   schema           VARCHAR2,
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 79–4 ANALYZE_SCHEMA Procedure Parameters

Parameter	Description
schema	Name of the schema.
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

Table 79–5 ANALYZE_SCHEMA Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this schema.

ANALYZE_DATABASE Procedure

This procedure runs the ANALYZE command on all the tables, clusters, and indexes in a database. Use this procedure to collect nonoptimizer statistics. For optimizer statistics, use the DBMS_STATS.GATHER_DATABASE_STATS procedure.

Syntax

```
DBMS_UTILITY.ANALYZE_DATABASE (
   method          VARCHAR2,
   estimate_rows   NUMBER   DEFAULT NULL,
   estimate_percent NUMBER  DEFAULT NULL,
   method_opt      VARCHAR2 DEFAULT NULL);
```

Parameters

Table 79–6 ANALYZE_DATABASE Procedure Parameters

Parameter	Description
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

Table 79–7 ANALYZE_DATABASE Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this database.

FORMAT_ERROR_STACK Function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK
  RETURN VARCHAR2;
```

Returns

This returns the error stack, up to 2000 bytes.

FORMAT_CALL_STACK Function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

Syntax

```
DBMS_UTILITY.FORMAT_CALL_STACK
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(format_call_stack,WNDS);
```

Returns

This returns the call stack, up to 2000 bytes.

IS_CLUSTER_DATABASE Function

This function finds out if this database is running in cluster database mode.

Syntax

```
DBMS_UTILITY.IS_CLUSTER_DATABASE
  RETURN BOOLEAN;
```

Returns

This function returns TRUE if this instance was started in cluster database mode; FALSE otherwise.

GET_TIME Function

This function finds out the current time in 100th's of a second. It is primarily useful for determining elapsed time.

Syntax

```
DBMS_UTILITY.GET_TIME
  RETURN NUMBER;
```

Returns

Time is the number of 100th's of a second from some arbitrary epoch.

GET_PARAMETER_VALUE Function

This function gets the value of specified `init.ora` parameter.

Syntax

```
DBMS_UTILITY.GET_PARAMETER_VALUE (
   parnam IN     VARCHAR2,
   intval IN OUT BINARY_INTEGER,
   strval IN OUT VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 79–8 GET_PARAMETER_VALUE Function Parameters

Parameter	Description
`parnam`	Parameter name.
`intval`	Value of an integer parameter or the value length of a string parameter.
`strval`	Value of a string parameter.

Returns

Table 79–9 GET_PARAMETER_VALUE Function Returns

Return	Description
partyp	Parameter type: 0 if parameter is an integer / boolean parameter 1 if parameter is a string / file parameter

Example

```
DECLARE
  parnam VARCHAR2(256);
  intval BINARY_INTEGER;
  strval VARCHAR2(256);
  partyp BINARY_INTEGER;
BEGIN
  partyp := dbms_utility.get_parameter_value('max_dump_file_size',
                                             intval, strval);
  dbms_output.put('parameter value is: ');
  IF partyp = 1 THEN
    dbms_output.put_line(strval);
  ELSE
    dbms_output.put_line(intval);
  END IF;
  IF partyp = 1 THEN
    dbms_output.put('parameter value length is: ');
    dbms_output.put_line(intval);
  END IF;
  dbms_output.put('parameter type is: ');
  IF partyp = 1 THEN
    dbms_output.put_line('string');
  ELSE
    dbms_output.put_line('integer');
  END IF;
END;
```

NAME_RESOLVE Procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

Syntax

```
DBMS_UTILITY.NAME_RESOLVE (
   name          IN  VARCHAR2,
   context       IN  NUMBER,
   schema        OUT VARCHAR2,
   part1         OUT VARCHAR2,
   part2         OUT VARCHAR2,
   dblink        OUT VARCHAR2,
   part1_type    OUT NUMBER,
   object_number OUT NUMBER);
```

Parameters

Table 79–10 NAME_RESOLVE Procedure Parameters

Parameter	Description
name	Name of the object. This can be of the form [[a.]b.]c[@d], where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the schema, part1, part2 and dblink OUT parameters are filled in. a, b and c may be delimited identifiers, and may contain NLS characters (single and multibyte).
context	Must be an integer between 0 and 8.
schema	Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.
part1	First part of the name. The type of this name is specified part1_type (synonym, procedure or package).
part2	If this is non-NULL, then this is a procedure name within the package indicated by part1.
dblink	If this is non-NULL, then a database link was either specified as part of name or name was a synonym which resolved to something with a database link. In this later case, part1_type indicates a synonym.

Table 79–10 NAME_RESOLVE Procedure Parameters

Parameter	Description
part1_type	Type of part1 is: 5 - synonym 7 - procedure (top level) 8 - function (top level) 9 - package If a synonym, then it means that name is a synonym that translates to something with a database link. In this case, if further name translation is desired, then you must call the DBMS_UTILITY.NAME_RESOLVE procedure on this remote node.
object_number	Object identifier

Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

NAME_TOKENIZE Procedure

This procedure calls the parser to parse the given name as "a [. b [. c]][@ dblink]". It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

Syntax

```
DBMS_UTILITY.NAME_TOKENIZE (
   name    IN  VARCHAR2,
   a       OUT VARCHAR2,
   b       OUT VARCHAR2,
   c       OUT VARCHAR2,
   dblink  OUT VARCHAR2,
   nextpos OUT BINARY_INTEGER);
```

Parameters

For each of a, b, c, dblink, tell where the following token starts in anext, bnext, cnext, dnext respectively.

COMMA_TO_TABLE Procedure

This procedure converts a comma-delimited list of names into a PL/SQL table of names. This uses NAME_TOKENIZE to figure out what are names and what are commas.

Syntax

```
DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT UNCL_ARRAY);
```

Parameters

Table 79–11 COMMA_TO_TABLE Procedure Parameters

Parameter	Description
list	Comma separated list of tables.
tablen	Number of tables in the PL/SQL table.
tab	PL/SQL table which contains list of table names.

Returns

A PL/SQL table is returned, with values 1..n and n+1 is null.

Usage Notes

The list must be a non-empty comma-delimited list: Anything other than a comma-delimited list is rejected. Commas inside double quotes do not count.

Entries in the comma-delimited list cannot include multibyte characters such as hyphens (-).

The values in tab are cut from the original list, with no transformations.

TABLE_TO_COMMA Procedure

This procedure converts a PL/SQL table of names into a comma-delimited list of names. This takes a PL/SQL table, 1..n, terminated with n+1 null.

Syntax

```
DBMS_UTILITY.TABLE_TO_COMMA (
```

```
   tab    IN  UNCL_ARRAY,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);
```

Parameters

Table 79–12 TABLE_TO_COMMA Procedure Parameters

Parameter	Description
tab	PL/SQL table which contains list of table names.
tablen	Number of tables in the PL/SQL table.
list	Comma separated list of tables.

Returns

Returns a comma-delimited list and the number of elements found in the table.

PORT_STRING Function

This function returns a string that identifies the operating system and the TWO TASK PROTOCOL version of the database. For example, "VAX/VMX-7.1.0.0"

The maximum length is port-specific.

Syntax

```
DBMS_UTILITY.PORT_STRING
   RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

DB_VERSION Procedure

This procedure returns version information for the database.

Syntax

```
DBMS_UTILITY.DB_VERSION (
   version       OUT VARCHAR2,
   compatibility OUT VARCHAR2);
```

Parameters

Table 79–13 DB_VERSION Procedure Parameters

Parameter	Description
version	A string which represents the internal software version of the database (for example, 7.1.0.0.0). The length of this string is variable and is determined by the database version.
compatibility	The compatibility setting of the database determined by the "compatible" init.ora parameter. If the parameter is not specified in the init.ora file, then NULL is returned.

MAKE_DATA_BLOCK_ADDRESS Function

This function creates a data block address given a file number and a block number. A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (
   file  NUMBER,
   block NUMBER)
  RETURN NUMBER;
```

Parameters

Table 79–14 MAKE_DATA_BLOCK_ADDRESS Function Parameters

Parameter	Description
file	File that contains the block.
block	Offset of the block within the file in terms of block increments.

Pragmas

```
pragma restrict_references(make_data_block_address, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 79–15 MAKE_DATA_BLOCK_ADDRESS Function Returns

Returns	Description
dba	Data block address.

DATA_BLOCK_ADDRESS_FILE Function

This function gets the file number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 79–16 DATA_BLOCK_ADDRESS_FILE Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references(data_block_address_file, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 79–17 DATA_BLOCK_ADDRESS_FILE Function Returns

Returns	Description
file	File that contains the block.

DATA_BLOCK_ADDRESS_BLOCK Function

This function gets the block number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (
   dba NUMBER)
```

```
  RETURN NUMBER;
```

Parameters

Table 79–18 DATA_BLOCK_ADDRESS_BLOCK Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 79–19 DATA_BLOCK_ADDRESS_BLOCK Function Returns

Returns	Description
block	Block offset of the block.

GET_HASH_VALUE Function

This function computes a hash value for the given string.

Syntax

```
DBMS_UTILITY.GET_HASH_VALUE (
   name      VARCHAR2,
   base      NUMBER,
   hash_size NUMBER)
  RETURN NUMBER;
```

Parameters

Table 79–20 GET_HASH_VALUE Function Parameters

Parameter	Description
name	String to be hashed.
base	Base value for the returned hash value to start at.
hash_size	Desired size of the hash table.

Pragmas

```
pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);
```

Returns

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the `hash_size` value. Using a power of 2 for the `hash_size` parameter works best.

ANALYZE_PART_OBJECT Procedure

This procedure is equivalent to SQL:

```
"ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>]
[<command_opt>] [<sample_clause>]
```

For each partition of the object, run in parallel using job queues.

Syntax

```
DBMS_UTILITY.ANALYZE_PART_OBJECT (
   schema        IN VARCHAR2 DEFAULT NULL,
   object_name   IN VARCHAR2 DEFAULT NULL,
   object_type   IN CHAR     DEFAULT 'T',
   command_type  IN CHAR     DEFAULT 'E',
   command_opt   IN VARCHAR2 DEFAULT NULL,
   sample_clause IN VARCHAR2 DEFAULT 'SAMPLE 5 PERCENT');
```

Parameters

Table 79–21 ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
`schema`	Schema of the `object_name`.
`object_name`	Name of object to be analyzed, must be partitioned.
`object_type`	Type of object, must be `T` (table) or `I` (index).

Table 79–21 ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
command_type	Must be one of the following: C (compute statistics) E (estimate statistics) D (delete statistics) V (validate structure)
command_opt	Other options for the command type. For C, E it can be FOR table, FOR all LOCAL indexes, FOR all columns or combination of some of the 'for' options of analyze statistics (table). For V, it can be CASCADE when object_type is T.
sample_clause	The sample clause to use when command_type is 'E'.

EXEC_DDL_STATEMENT Procedure

This procedure executes the DDL statement in parse_string.

Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (
   parse_string IN VARCHAR2);
```

Parameters

Table 79–22 EXEC_DDL_STATEMENT Procedure Parameters

Parameter	Description
parse_string	DDL statement to be executed.

CURRENT_INSTANCE Function

This function returns the current connected instance number. It returns NULL when connected instance is down.

Syntax

```
DBMS_UTILITY.CURRENT_INSTANCE
   RETURN NUMBER;
```

ACTIVE_INSTANCES Procedure

Syntax

```
DBMS_UTILITY.ACTIVE_INSTANCE (
   instance_table   OUT INSTANCE_TABLE,
   instance_count   OUT NUMBER);
```

Parameters

Table 79–23 ACTIVE_INSTANCES Procedure Parameters

Procedure	Description
`instance_table`	Contains a list of the active instance numbers and names. When no instance is up, the list is empty.
`instance_count`	Number of active instances.

80

DBMS_WM

This chapter describes how to use the DBMS_WM package, the programming interface to Oracle Database Workspace Manager (often referred to as Workspace Manager) to work with long transactions.

Workspace management refers to the ability of the database to hold different versions of the same record (that is, row) in one or more workspaces. Users of the database can then change these versions independently.

> **See Also:** *Oracle9i Application Developer's Guide - Workspace Manager* for detailed conceptual and usage information about Workspace Manager. That manual also includes the reference information found in this chapter.

This chapter discusses the following topics:

Summary of DBMS_WM Subprograms

Table 80–1 DBMS_WM Subprograms

Subprogram	Description
AlterSavepoint Procedure on page 80-6	Modifies the description of a savepoint.
AlterWorkspace Procedure on page 80-7	Modifies the description of a workspace.
BeginDDL Procedure on page 80-8	Starts a DDL (data definition language) session for a specified table.
BeginResolve Procedure on page 80-9	Starts a conflict resolution session.
CommitDDL Procedure on page 80-10	Commits DDL (data definition language) changes made during a DDL session for a specified table, and ends the DDL session.
CommitResolve Procedure on page 80-12	Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since BeginResolve was executed.
CompressWorkspace Procedure on page 80-13	Deletes removable savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace.
CompressWorkspaceTree Procedure on page 80-16	Deletes removable savepoints in a workspace and all its descendant workspaces. It also minimizes the Workspace Manager metadata structures for the affected workspaces, and eliminates any redundant data that might arise from the deletion of the savepoints.
CopyForUpdate Procedure on page 80-17	Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified.
CreateSavepoint Procedure on page 80-19	Creates a savepoint for the current version.
CreateWorkspace Procedure on page 80-20	Creates a new workspace in the database.
DeleteSavepoint Procedure on page 80-22	Deletes a savepoint and associated rows in version-enabled tables.
DisableVersioning Procedure on page 80-24	Deletes all support structures that were created to enable the table to support versioned rows.
DropReplicationSupport Procedure on page 80-26	Deletes replication support objects that had been created by the GenerateReplicationSupport procedure.

Table 80–1 DBMS_WM Subprograms (Cont.)

Subprogram	Description
EnableVersioning Procedure on page 80-27	Version-enables a table, creating the necessary structures to enable the table to support multiple versions of rows.
FreezeWorkspace Procedure on page 80-30	Restricts access to a workspace and the ability of users to make changes in the workspace.
GenerateReplicationSupport Procedure on page 80-32	Creates necessary structures for multimaster replication of Workspace Manager objects, and starts the master activity for the newly created master group.
GetConflictWorkspace Function on page 80-34	Returns the name of the workspace on which the session has performed the SetConflictWorkspace procedure.
GetDiffVersions Function on page 80-35	Returns the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions operation.
GetLockMode Function on page 80-35	Returns the locking mode for the current session, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.
GetMultiWorkspaces Function on page 80-36	Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.
GetOpContext Function on page 80-37	Returns the context of the current operation for the current session.
GetPrivs Function on page 80-38	Returns a comma-delimited list of all privileges that the current user has for the specified workspace.
GetSessionInfo Procedure on page 80-38	Retrieves information about the current workspace and session context.
GetWorkspace Function on page 80-40	Returns the current workspace for the session.
GotoDate Procedure on page 80-41	Goes to a point at or near the specified date and time in the current workspace.
GotoSavepoint Procedure on page 80-42	Goes to the specified savepoint in the current workspace.
GotoWorkspace Procedure on page 80-43	Moves the current session to the specified workspace.
GrantSystemPriv Procedure on page 80-44	Grants system-level privileges (not restricted to a particular workspace) to users and roles. The `grant_option` parameter enables the grantee to then grant the specified privileges to other users and roles.

Table 80–1 DBMS_WM Subprograms (Cont.)

Subprogram	Description
GrantWorkspacePriv Procedure on page 80-46	Grants workspace-level privileges to users and roles. The `grant_option` parameter enables the grantee to then grant the specified privileges to other users and roles.
IsWorkspaceOccupied Function on page 80-48	Checks whether or not a workspace has any active sessions.
LockRows Procedure on page 80-49	Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.
MergeTable Procedure on page 80-50	Applies changes to a table (all rows or as specified in the `WHERE` clause) in a workspace to its parent workspace.
MergeWorkspace Procedure on page 80-52	Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.
RecoverAllMigratingTables Procedure on page 80-54	Attempts to complete the migration process on all tables that were left in an inconsistent state after the Workspace Manager migration procedure failed.
RecoverMigratingTable Procedure on page 80-55	Attempts to complete the migration process on a table that was left in an inconsistent state after the Workspace Manager migration procedure failed.
RefreshTable Procedure on page 80-57	Applies to a workspace all changes made to a table (all rows or as specified in the `WHERE` clause) in its parent workspace.
RefreshWorkspace Procedure on page 80-58	Applies to a workspace all changes made in its parent workspace.
RelocateWriterSite Procedure on page 80-59	Makes one of the nonwriter sites the new writer site in a Workspace Manager replication environment. (The old writer site becomes one of the nonwriter sites.)
RemoveWorkspace Procedure on page 80-61	Discards all row versions associated with a workspace and deletes the workspace.
RemoveWorkspaceTree Procedure on page 80-62	Discards all row versions associated with a workspace and its descendant workspaces, and deletes the affected workspaces.
ResolveConflicts Procedure on page 80-63	Resolves conflicts between workspaces.
RevokeSystemPriv Procedure on page 80-65	Revokes (removes) system-level privileges from users and roles.
RevokeWorkspacePriv Procedure on page 80-67	Revokes (removes) workspace-level privileges from users and roles for a specified workspace.

Table 80–1 DBMS_WM Subprograms (Cont.)

Subprogram	Description
RollbackDDL Procedure on page 80-68	Rolls back (cancels) DDL changes made during a DDL session for a specified table, and ends the DDL session.
RollbackResolve Procedure on page 80-69	Quits a conflict resolution session and discards all changes in the workspace since BeginResolve was executed.
RollbackTable Procedure on page 80-70	Discards all changes made in the workspace to a specified table (all rows or as specified in the `WHERE` clause).
RollbackToSP Procedure on page 80-72	Discards all changes made in a workspace to version-enabled tables since a specified savepoint.
RollbackWorkspace Procedure on page 80-73	Discards all changes made in the workspace to version-enabled tables.
SetConflictWorkspace Procedure on page 80-74	Determine whether or not conflicts exist between a workspace and its parent.
SetDiffVersions Procedure on page 80-75	Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It modifies the contents of the differences views that describe these differences.
SetLockingOFF Procedure on page 80-77	Disables Workspace Manager locking for the current session.
SetLockingON Procedure on page 80-78	Enables Workspace Manager locking for the current session.
SetMultiWorkspaces Procedure on page 80-79	Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.
SetWoOverwriteOFF Procedure on page 80-80	Disables the `VIEW_WO_OVERWRITE` history option that had been enabled by the `EnableVersioning` or `SetWoOverwriteON` procedure, changing the option to `VIEW_W_OVERWRITE` (with overwrite).
SetWoOverwriteON Procedure on page 80-81	Enables the `VIEW_WO_OVERWRITE` history option that had been disabled by the `SetWoOverwriteOFF` procedure.
SetWorkspaceLockModeOFF Procedure on page 80-82	Disables Workspace Manager locking for the specified workspace.
SetWorkspaceLockModeON Procedure on page 80-83	Enables Workspace Manager locking for the specified workspace.
SynchronizeSite Procedure on page 80-85	Brings the local site (the old writer site) up to date in the Workspace Manager replication environment after the writer site was moved using the RelocateWriterSite procedure.

Table 80–1 DBMS_WM Subprograms (Cont.)

Subprogram	Description
UnfreezeWorkspace Procedure on page 80-86	Enables access and changes to a workspace, reversing the effect of `FreezeWorkspace`.
UnlockRows Procedure on page 80-87	Enables access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Note: Most Workspace Manager subprograms are procedures, but a few are functions. Most functions have names starting with *Get* (such as the GetConflictWorkspace Function and GetWorkspace Function).

In this chapter, the term *procedures* is often used to refer generally to both procedures and functions.

AlterSavepoint Procedure

Modifies the description of a savepoint.

Syntax

```
DBMS_WM.AlterSavepoint(
   workspace      IN VARCHAR2,
   sp_name        IN VARCHAR2,
   sp_description IN VARCHAR2);
```

Parameters

Table 80–2 AlterSavepoint Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace in which the savepoint was created. The name is case sensitive.
`sp_name`	Name of the savepoint. The name is case sensitive.
`sp_description`	Description of the savepoint.

Usage Notes

To see the current description of the savepoint, examine the DESCRIPTION column value for the savepoint in the ALL_WORKSPACE_SAVEPOINTS metadata view, which is described in *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if the user is not the workspace owner or savepoint owner or does not have the WM_ADMIN_ROLE role.

Examples

The following example modifies the description of savepoint SP1 in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.AlterSavepoint ('NEWWORKSPACE', 'SP1', 'First set of changes for
scenario');
```

AlterWorkspace Procedure

Modifies the description of a workspace.

Syntax

```
DBMS_WM.AlterWorkspace(
   workspace              IN VARCHAR2,
   workspace_description  IN VARCHAR2);
```

Parameters

Table 80–3 AlterWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
workspace_description	Description of the workspace.

Usage Notes

To see the current description of the workspace, examine the DESCRIPTION column value for the savepoint in the ALL_WORKSPACES metadata view, which is described in *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if the user is not the workspace owner or does not have the WM_ADMIN_ROLE role.

Examples

The following example modifies the description of the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.AlterWorkspace ('NEWWORKSPACE', 'Testing proposed scenario B');
```

BeginDDL Procedure

Starts a DDL (data definition language) session for a specified table.

Syntax

```
DBMS_WM.BeginDDL(
   table_name  IN VARCHAR2);
```

Parameters

Table 80–4 BeginDDL Procedure Parameters

Parameter	Description
table_name	Name of the version-enabled table. The name is not case sensitive.

Usage Notes

This procedure starts a DDL session, and it creates a special table whose name is the same as table_name but with *_LTS* added to the table name. After calling this procedure, you can perform one or more DDL operations on the table or any indexes or triggers that are based on the table, and then call either the CommitDDL Procedure or RollbackDDL Procedure.

In addition to creating the special <*table-name*>*_LTS* table, the procedure creates other objects:

- The <*table-name*>*_LTS* table has the same triggers, columns, and indexes as the <*table-name*> table.
- For each parent table with which the <*table-name*> table has a referential integrity constraint, the same constraint is defined for the <*table-name*>*_LTS* table.
- Triggers, columns, and referential integrity constraints on the <*table-name*>*_LTS* table have the same names as the corresponding ones on the <*table-name*> table.
- For each index on the <*table-name*> table, the corresponding index on the <*table-name*>*_LTS* table has a name in the form <*index-name*>*_LTS*.

- The primary key constraint on the <*table-name*>*_LTS* table has a name in the form <*primary-key*>*_LTS*.

For detailed information about performing DDL operations related to version-enabled tables and about DDL operations on version-enabled tables in an Oracle replication environment, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- `table_name` does not exist or is not version-enabled.
- The user does not have the `CREATE TABLE` privilege.
- An open DDL session exists for `table_name`. (That is, the BeginDDL procedure has already been called specifying this table, and the CommitDDL Procedure or RollbackDDL Procedure has not been called specifying this table.)

Examples

The following example begins a DDL session, adds a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the special table named `COLA_MARKETING_BUDGET_LTS`, and ends the DDL session by committing the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
EXECUTE DBMS_WM.CommitDDL('COLA_MARKETING_BUDGET');
```

BeginResolve Procedure

Starts a conflict resolution session.

Syntax

```
DBMS_WM.BeginResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–5 BeginResolve Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.

Usage Notes

This procedure starts a conflict resolution session. While this procedure is executing, the workspace is frozen in `1WRITER` mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

After calling this procedure, you can execute the ResolveConflicts Procedure as needed for various tables that have conflicts, and then call either the CommitResolve Procedure or RollbackResolve Procedure. For more information about conflict resolution, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The user executing the BeginResolve Procedure does not have the privilege to access `workspace` and its parent workspace.

Examples

The following example starts a conflict resolution session in `Workspace1`.

```
EXECUTE  DBMS_WM.BeginResolve ('Workspace1');
```

CommitDDL Procedure

Commits DDL (data definition language) changes made during a DDL session for a specified table, and ends the DDL session.

Syntax

```
DBMS_WM.CommitDDL(
   table_name           IN VARCHAR2
   [, ignore_last_error IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–6 CommitDDL Procedure Parameters

Parameter	Description
`table_name`	Name of the version-enabled table. The name is not case sensitive.

Table 80–6 CommitDDL Procedure Parameters (Cont.)

Parameter	Description
ignore_ last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the previous call to the CommitDDL procedure. Information about the last error is stored in the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views, which are described in *Oracle9i Application Developer's Guide - Workspace Manager*. (See the Usage Notes for more information.) FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the CommitDDL procedure.

Usage Notes

This procedure commits changes that were made to a version-enabled table and to any indexes, triggers, and referential integrity constraints based on the version-enabled table during a DDL session. It also deletes the special <*table-name*>_*LTS* table that had been created by the BeginDDL Procedure.

For detailed information about performing DDL operations related to version-enabled tables and about DDL operations on version-enabled tables in an Oracle replication environment, see *Oracle9i Application Developer's Guide - Workspace Manager*.

If a call to the CommitDDL procedure fails, the table is left in an inconsistent state. If this occurs, you should try to fix the cause of the error. Examine the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views to see the SQL statement and error message. For example, the CommitDDL procedure might have failed because the tablespace was not large enough to add a column. Fix the cause of the error, and then call the CommitDDL procedure again with the default ignore_last_error parameter value of FALSE. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the CommitDDL procedure with the ignore_last_error parameter value of TRUE. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if one or more of the following apply:

- table_name does not exist or is not version-enabled.
- The user does not have the CREATE TABLE privilege.

- An open DDL session does not exist for `table_name`. (That is, the BeginDDL Procedure has not been called specifying this table, or the CommitDDL Procedure or RollbackDDL Procedure was already called specifying this table.)

Some invalid DDL operations also cause an exception when CommitDDL procedure is called. (See *Oracle9i Application Developer's Guide - Workspace Manager* for information about DDL operations that are supported.)

Examples

The following example begins a DDL session, adds a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the special table named `COLA_MARKETING_BUDGET_LTS`, and ends the DDL session by committing the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
EXECUTE DBMS_WM.CommitDDL('COLA_MARKETING_BUDGET');
```

CommitResolve Procedure

Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since the BeginResolve Procedure was executed.

Syntax

```
DBMS_WM.CommitResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–7 CommitResolve Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.

Usage Notes

This procedure ends the current conflict resolution session (started by the BeginResolve Procedure), and saves all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with the RollbackResolve Procedure, which discards all changes.

For more information about conflict resolution, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The procedure was called by a user that does not have the `WM_ADMIN_ROLE` role or that did not execute the BeginResolve Procedure on `workspace`.

Examples

The following example ends the conflict resolution session in `Workspace1` and saves all changes.

```
EXECUTE  DBMS_WM.CommitResolve ('Workspace1');
```

CompressWorkspace Procedure

Deletes removable savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace. (*Removable savepoints* are explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)

Syntax

```
DBMS_WM.CompressWorkspace(
   workspace                   IN VARCHAR2,
   compress_view_wo_overwrite  IN BOOLEAN
   [, firstSP                  IN VARCHAR2 DEFAULT NULL
   [, secondSP                 IN VARCHAR2 DEFAULT NULL] ]
   [, auto_commit              IN BOOLEAN DEFAULT TRUE]);
```

or

```
DBMS_WM.CompressWorkspace(
   workspace                   IN VARCHAR2
   [, firstSP                  IN VARCHAR2 DEFAULT NULL
   [, secondSP                 IN VARCHAR2 DEFAULT NULL] ]
   [, auto_commit              IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–8 CompressWorkspace Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.

Table 80–8 CompressWorkspace Procedure Parameters (Cont.)

Parameter	Description
compress_ view_wo_ overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information between the affected savepoints to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE causes history information (between the affected savepoints) for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.) FALSE is assumed if the procedure format without this parameter is used.
firstSP	First savepoint. Savepoint names are case sensitive. If only workspace and firstSP are specified, all removable savepoints between workspace creation and firstSP (but not including firstSP) are deleted. If workspace, firstSP, and secondSP are specified, all removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to secondSP (but not including secondSP) are deleted. If only workspace is specified (no savepoints), all removable savepoints in the workspace are deleted.
secondSP	Second savepoint. All removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to secondSP (but not including secondSP) are deleted. However, if secondSP is LATEST, all removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to the end of the workspace are deleted. Savepoint names are case sensitive.
auto_ commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

You can compress a workspace when the explicit savepoints (all or some of them) in the workspace are no longer needed. The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in NO_ACCESS mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

A workspace cannot be compressed if there are any sessions in the workspace (except for the LIVE workspace), or if any user has executed a GotoDate Procedure operation or a GotoSavepoint Procedure operation specifying a savepoint in the workspace.

If the procedure format without the compress_view_wo_overwrite parameter is used, a value of FALSE is assumed for the parameter.

For information about VIEW_WO_OVERWRITE and other history options, see the information about the EnableVersioning Procedure.

An exception is raised if the user does not have the privilege to access and merge changes in workspace.

To compress a workspace and all its descendant workspaces, use the CompressWorkspaceTree Procedure.

Examples

The following example compresses NEWWORKSPACE.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE');
```

The following example compresses NEWWORKSPACE, deleting all explicit savepoints between the creation of the workspace and the savepoint SP1.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1');
```

The following example compresses NEWWORKSPACE, deleting the explicit savepoint SP1 and all explicit savepoints up to but not including SP2.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1', 'SP2');
```

The following example compresses B_focus_1, accepts the default values for the firstSP and secondSP parameters (that is, deletes all explicit savepoints), and specifies FALSE for the auto_commit parameter.

```
EXECUTE DBMS_WM.CompressWorkspace ('B_focus_1', auto_commit => FALSE);
```

CompressWorkspaceTree Procedure

Deletes removable savepoints in a workspace and all its descendant workspaces. (*Removable savepoints* are explained in *Oracle9i Application Developer's Guide - Workspace Manager*.) It also minimizes the Workspace Manager metadata structures for the affected workspaces, and eliminates any redundant data that might arise from the deletion of the savepoints.

Syntax

```
DBMS_WM.CompressWorkspaceTree(
   workspace                      IN VARCHAR2
   [, compress_view_wo_overwrite  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit                 IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–9 CompressWorkspaceTree Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
compress_view_wo_overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE (the default) causes history information for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.)
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

You can compress a workspace and all its descendant workspaces when the explicit savepoints in the affected workspaces are no longer needed (for example, if you will not need to go to or roll back to any of these savepoints). For an explanation of

database workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.

The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in NO_ACCESS mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

A workspace cannot be compressed if there are any sessions in the workspace (except for the LIVE workspace), or if any user has executed a GotoDate Procedure operation or a GotoSavepoint Procedure operation specifying a savepoint in the workspace.

An exception is raised if the user does not have the privilege to access and merge changes in workspace.

If the CompressWorkspaceTree operation fails in any affected workspace, the entire operation is rolled back, and no workspaces are compressed.

To compress a single workspace (deleting all explicit savepoints or just some of them), use the CompressWorkspace Procedure.

Examples

The following example compresses NEWWORKSPACE and all its descendant workspaces.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('NEWWORKSPACE');
```

The following example compresses NEWWORKSPACE and all its descendant workspaces, accepts the default value for the compress_view_wo_overwrite parameter, and specifies FALSE for the auto_commit parameter.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('B_focus_1', auto_commit => FALSE);
```

CopyForUpdate Procedure

Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified. Use this procedure only if a version-enabled table has any LOB columns.

Syntax

```
DBMS_WM.CopyForUpdate(
   table_name       IN VARCHAR2,
   [, where_clause  IN VARCHAR2 DEFAULT '']);
```

Parameters

Table 80–10 CopyForUpdate Procedure Parameters

Parameter	Description
table_name	Name of the table containing one or more LOB columns. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows affected. Example: 'department_id = 20'
	Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery.
	If where_clause is not specified, all rows in table_name are affected.

Usage Notes

This procedure is intended for use only with version-enabled tables containing one or more large object (LOB) columns. The CopyForUpdate procedure must be used because updates performed using the DBMS_LOB package do not fire INSTEAD OF triggers on the versioning views. Workspace Manager creates INSTEAD OF triggers on the versioning views to implement the copy-on-write semantics. (For non-LOB columns, you can directly perform the update operation, and the triggers work.)

Examples

The following example updates the SOURCE_CLOB column of TABLE1 for the document with DOC_ID = 1.

```
Declare
  clob_var
Begin
   /* This procedure copies the LOB columns if necessary, that is,
      if the row with doc_id = 1 has not been versioned in the
      current version */
   dbms_wm.copyForUpdate('table1', 'doc_id = 1');

   select source_clob into clob_var
   from   table1
```

```
    where  doc_id = 1 for update;

    dbms_lob.write(clob_var,<amount>, <offset>, buff);

End;
```

CreateSavepoint Procedure

Creates a savepoint for the current version.

Syntax

```
DBMS_WM.CreateSavepoint(
   workspace       IN VARCHAR2,
   savepoint_name  IN VARCHAR2
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–11 CreateSavepoint Procedure Parameters

Parameter	Description
workspace	Name of the workspace in which to create the savepoint. The name is case sensitive.
savepoint_name	Name of the savepoint to be created. The name is case sensitive.
description	Description of the savepoint to be created.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

There are no explicit privileges associated with savepoints; any user who can access a workspace can create a savepoint in the workspace.

This procedure can be performed while there are users in the workspace; there can be open database transactions.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- The user is not in the latest version in the workspace (for example, if the user has called the GotoDate Procedure).
- workspace does not exist.
- savepoint_name already exists.
- The user does not have the privilege to go to the specified workspace.

Examples

The following example creates a savepoint named Savepoint1 in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.CreateSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

CreateWorkspace Procedure

Creates a new workspace in the database.

Syntax

```
DBMS_WM.CreateWorkspace(
   workspace       IN VARCHAR2
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

or

```
DBMS_WM.CreateWorkspace(
   workspace       IN VARCHAR2,
   isrefreshed     IN BOOLEAN
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–12 CreateWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive, and it must be unique (no other workspace of the same name).
isrefreshed	A Boolean value (TRUE or FALSE). TRUE causes the workspace to be continually refreshed. In a **continually refreshed workspace**, changes made in the parent workspace are automatically applied to the workspace after a merge or rollback operation in the parent workspace. That is, you do not need to call the RefreshWorkspace Procedure to apply the changes. See the Usage Notes for more information about continually refreshed workspaces. FALSE causes the workspace not to be continually refreshed. To refresh the workspace, you must call the RefreshWorkspace Procedure. If you use the syntax without the isrefreshed parameter, the workspace is not continually refreshed.
description	Description of the workspace.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

The new workspace is a child of the current workspace. If the session has not explicitly entered a workspace, it is in the LIVE database workspace, and the new workspace is a child of the LIVE workspace. For an explanation of database workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An implicit savepoint is created in the current version of the current workspace. (The current version does not have to be the latest version in the current workspace.) For an explanation of savepoints (explicit and implicit), see *Oracle9i Application Developer's Guide - Workspace Manager*.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

This procedure does not implicitly go to the workspace created. To go to the workspace, use the GotoWorkspace Procedure.

The following rules apply to continually refreshed workspaces (`isrefreshed` value of `TRUE`):

- A continually refreshed workspace must be created as a child of the `LIVE` workspace.
- A continually refreshed workspace must be a leaf workspace (that is, have no child workspaces).
- The session must be on the latest version in order to create a continually refreshed workspace.
- You cannot turn off locking using the SetLockingOFF Procedure or SetWorkspaceLockModeOFF Procedure for a continually refreshed workspace.

An exception is raised if one or more of the following apply:

- `workspace` already exists.
- The user does not have the privilege to create a workspace.

Examples

The following example creates a workspace named `NEWWORKSPACE` in the database.

```
EXECUTE DBMS_WM.CreateWorkspace ('NEWWORKSPACE');
```

DeleteSavepoint Procedure

Deletes a savepoint and associated rows in version-enabled tables.

Syntax

```
DBMS_WM.DeleteSavepoint(
   workspace                      IN VARCHAR2,
   savepoint_name                 IN VARCHAR2)
   [, compress_view_wo_overwrite  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit                 IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–13 DeleteSavepoint Procedure Parameters

Parameter	Description
workspace	Name of the workspace in which the savepoint was created. The name is case sensitive.
savepoint_name	Name of the savepoint to be deleted. The name is case sensitive.
compress_view_wo_overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE (the default) causes history information for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.)
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

You can delete a savepoint when it is no longer needed (for example, you will not need to go to it or roll back to it).

Deleting a savepoint is useful for the following reasons:

- You can reuse a savepoint name after it is deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in NO_ACCESS mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

To delete a savepoint, you must have the WM_ADMIN_ROLE role or be the owner of the workspace or the savepoint.

This procedure cannot be executed if there are any sessions with an open database transaction, or if any user has executed a GotoDate Procedure operation or a GotoSavepoint Procedure operation specifying a savepoint in the workspace.

An exception is raised if one or more of the following apply:

- One or more sessions are already in `workspace` (unless the workspace is `LIVE`).
- `workspace` does not exist.
- `savepoint_name` does not exist.
- `savepoint_name` is not a removable savepoint. (Removable savepoints are explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)
- The user does not have the privilege to go to the specified workspace.

Examples

The following example deletes a savepoint named `Savepoint1` in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.DeleteSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

DisableVersioning Procedure

Deletes all support structures that were created to enable the table to support versioned rows.

Syntax

```
DBMS_WM.DisableVersioning(
   table_name           IN VARCHAR2
   [, force             IN BOOLEAN DEFAULT FALSE]
   [, ignore_last_error IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–14 DisableVersioning Procedure Parameters

Parameter	Description
`table_name`	Name of the table, or a comma-delimited list of names of tables related by multilevel referential integrity constraints. (Multilevel referential integrity constraints are explained in *Oracle9i Application Developer's Guide - Workspace Manager*.) Table names are not case sensitive.

Table 80–14 DisableVersioning Procedure Parameters (Cont.)

Parameter	Description
force	A Boolean value (TRUE or FALSE). TRUE forces all data in workspaces other than LIVE to be discarded before versioning is disabled. FALSE (the default) prevents versioning from being disabled if table_name was modified in any workspace other than LIVE and if the workspace that modified table_name still exists.
ignore_last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the previous call to the DisableVersioning procedure. Information about the last error is stored in the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views, which are described in *Oracle9i Application Developer's Guide - Workspace Manager*. (See the Usage Notes for more information.) FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the DisableVersioning procedure.

Usage Notes

This procedure is used to reverse the effect of the EnableVersioning Procedure. It deletes the Workspace Manager infrastructure (support structures) for versioning of rows, but does not affect any user data in the LIVE workspace. The workspace hierarchy and any savepoints still exist, but all rows are the same as in the LIVE workspace. (If there are multiple versions in the LIVE workspace of a row in the table for which versioning is disabled, only the most recent version of the row is kept.)

If a call to the DisableVersioning procedure fails, the table is left in an inconsistent state. If this occurs, you should try to fix the cause of the error (examine the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views to see the SQL statement and error message), and then call the DisableVersioning procedure again with the default ignore_last_error parameter value of FALSE. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the DisableVersioning procedure with the ignore_last_error parameter value of TRUE. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

Some causes for the failure of the DisableVersioning procedure include the following:

- The table contains much data in workspaces and the size of the undo tablespace required for the DisableVersioning procedure is not sufficient.
- A compilation error occurred while transferring user-defined triggers from the version-enabled table to the version-disabled table.

The DisableVersioning operation fails if the `force` value is `FALSE` and any of the following apply:

- The table is being modified by any user in any workspace other than the `LIVE` workspace.
- There are versioned rows of the table in any workspace other than the `LIVE` workspace.

Only the owner of a table or a user with the `WM_ADMIN_ROLE` role can disable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

An exception is raised if the table is not version-enabled.

If you want to disable versioning on a table in an Oracle replication environment, see *Oracle9i Application Developer's Guide - Workspace Manager* for guidelines and other information.

Examples

The following example disables the `EMPLOYEE` table for versioning.

```
EXECUTE DBMS_WM.DisableVersioning ('employee');
```

The following example disables the `EMPLOYEE` table for versioning and ignores the last error that occurred during the previous call to the DisableVersioning procedure.

```
EXECUTE DBMS_WM.DisableVersioning ('employee', ignore_last_error => true);
```

The following example disables the `EMPLOYEE`, `DEPARTMENT`, and `LOCATION` tables (which have multilevel referential integrity constraints) for versioning.

```
EXECUTE DBMS_WM.DisableVersioning('employee,department,location');
```

DropReplicationSupport Procedure

Deletes replication support objects that had been created by the GenerateReplicationSupport Procedure.

Syntax

```
DBMS_WM.DropReplicationSupport();
```

Parameters

None.

Usage Notes

To use this procedure, you must understand how replication applies to Workspace Manager objects, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*. You must also understand the major Oracle replication concepts and techniques, which are documented in *Oracle9i Replication* and *Oracle9i Replication Management API Reference*.

You must execute this procedure as the replication administrator user at the writer site.

This procedure drops replication support for any version-enabled tables at the nonwriter sites; however, it does not version-disable any version-enabled tables.

Examples

The following example drops replication support that had previously been enabled using the GenerateReplicationSupport Procedure.

```
DBMS_WM.DropReplicationSupport();
```

EnableVersioning Procedure

Version-enables a table, creating the necessary structures to enable the table to support multiple versions of rows.

Syntax

```
DBMS_WM.EnableVersioning(
   table_name  IN VARCHAR2
   [, hist     IN VARCHAR2 DEFAULT 'NONE']);
```

Parameters

Table 80–15 EnableVersioning Procedure Parameters

Parameter	Description
table_name	Name of the table, or a comma-delimited list of names of tables related by multilevel referential integrity constraints. (Multilevel referential integrity constraints are explained in *Oracle9i Application Developer's Guide - Workspace Manager*.) The length of a table name must not exceed 25 characters. The name is not case sensitive.
hist	History option, for tracking modifications to table_name. Must be one of the following values: NONE: No modifications to the table are tracked. (This is the default.) VIEW_W_OVERWRITE: The *with* overwrite (*W*_OVERWRITE) option. A view named <table_name>_HIST (described in *Oracle9i Application Developer's Guide - Workspace Manager*) is created to contain history information, but it will show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes. (The CREATETIME column of the <table_name>_HIST view contains only the time of the most recent update.) VIEW_WO_OVERWRITE: The *without* overwrite (*WO*_OVERWRITE) option. A view named <table_name>_HIST (described in *Oracle9i Application Developer's Guide - Workspace Manager*) is created to contain history information, and it will show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.

Usage Notes

The table that is being version-enabled must have a primary key defined.

Only the owner of a table or a user with the WM_ADMIN role can enable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

Tables owned by SYS cannot be version-enabled.

An exception is raised if one or more of the following apply:

- table_name is already version-enabled.

- table_name contains a list of tables and any of the tables has a referential integrity constraint with a table that is not in the list.

If the table is version-enabled with the VIEW_WO_OVERWRITE *hist* option specified, this option can later be disabled and re-enabled by calling the SetWoOverwriteOFF Procedure and SetWoOverwriteON Procedures.

The history option enables you to log and audit modifications.

The history option affects the behavior of the GotoDate Procedure. See the Usage Notes for that procedure.

If you want to version-enable a table in an Oracle replication environment, see *Oracle9i Application Developer's Guide - Workspace Manager* for guidelines and other information.

Current notes and restrictions include the following:

- If you have referential integrity constraints on version-enabled tables, note the considerations and restrictions in *Oracle9i Application Developer's Guide - Workspace Manager*.
- If you have triggers defined on version-enabled tables, note the considerations and restrictions in *Oracle9i Application Developer's Guide - Workspace Manager*.
- Constraints and privileges defined on the table are carried over to the version-enabled table.
- DDL operations on version-enabled tables are subject to the procedures and restrictions described in *Oracle9i Application Developer's Guide - Workspace Manager*.
- Index-organized tables cannot be version-enabled.
- Object tables cannot be version-enabled.
- A table with one or more columns of LONG data type cannot be version-enabled.
- A table with one or more nested table columns cannot be version-enabled.

Examples

The following example enables versioning on the EMPLOYEE table.

```
EXECUTE DBMS_WM.EnableVersioning('employee');
```

The following example enables versioning on the EMPLOYEE, DEPARTMENT, and LOCATION tables, which have multilevel referential integrity constraints.

```
EXECUTE DBMS_WM.EnableVersioning('employee,department,location');
```

FreezeWorkspace Procedure

Restricts access to a workspace and the ability of users to make changes in the workspace.

Syntax

```
DBMS_WM.FreezeWorkspace(
   workspace        IN VARCHAR2
   [, freezemode    IN VARCHAR2 DEFAULT 'NO_ACCESS']
   [, freezewriter  IN VARCHAR2 DEFAULT NULL]
   [, force         IN BOOLEAN DEFAULT FALSE]);
```

or

```
DBMS_WM.FreezeWorkspace(
   workspace        IN VARCHAR2,
   session_duration IN BOOLEAN
   [, freezemode    IN VARCHAR2 DEFAULT 'NO_ACCESS']
   [, freezewriter  IN VARCHAR2 DEFAULT NULL]
   [, force         IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–16 FreezeWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
session_ duration	A Boolean value (TRUE or FALSE). TRUE causes the workspace to be unfrozen when the session that called the FreezeWorkspace procedure disconnects from the database. This value is valid for all freeze modes. FALSE causes the workspace not to be unfrozen when the session that called the FreezeWorkspace procedure disconnects from the database.

Table 80–16 FreezeWorkspace Procedure Parameters (Cont.)

Parameter	Description
freezemode	Mode for the frozen workspace. Must be one of the following values: NO_ACCESS: No sessions are allowed in the workspace. (This is the default.) READ_ONLY: Sessions are allowed in the workspace, but no write operations (insert, update, delete) are allowed. 1WRITER: Sessions are allowed in the workspace, but only one user (see the freezewriter parameter) is allowed to perform write operations (insert, update, delete). 1WRITER_SESSION: Sessions are allowed in the workspace, but only the database session (as opposed to the database user) that called the FreezeWorkspace procedure is allowed to perform write operations (insert, update, delete). The workspace is unfrozen after the session that called the FreezeWorkspace procedure disconnects from the database. WM_ONLY: Only Workspace Manager operations are permitted. No sessions can directly modify data values or perform queries involving table data; however, child workspaces can be merged into the workspace, and savepoints can be created in the workspace.
freezewriter	The user that is allowed to make changes in the workspace. Can be specified only if freezemode is 1WRITER. The default is USER (the current user).
force	A Boolean value (TRUE or FALSE). TRUE forces the workspace to be frozen even if it is already frozen. For example, this value lets you freeze the workspace with a different freezemode parameter value without having first to call the UnfreezeWorkspace Procedure. FALSE (the default) prevents the workspace from being frozen if it is already frozen.

Usage Notes

If you specify the procedure syntax that does not include the session_duration parameter, it is equivalent to specifying FALSE for that parameter: that is, the workspace is not unfrozen when the session that called the FreezeWorkspace procedure disconnects from the database.

The operation fails if one or more of the following apply:

- workspace is already frozen (unless force is TRUE).

- Any sessions are in workspace and freezemode is NO_ACCESS (specified or defaulted).
- session_duration is FALSE and freezemode is 1WRITER_SESSION.

If freezemode is READ_ONLY or 1WRITER, the workspace cannot be frozen if there is an active database transaction.

You can freeze a workspace only if one or more of the following apply:

- You are the owner of the specified workspace.
- You have the WM_ADMIN_ROLE, the FREEZE_ANY_WORKSPACE privilege, or the FREEZE_WORKSPACE privilege for the specified workspace.

The LIVE workspace can be frozen only if freezemode is READ_ONLY or 1WRITER.

To reverse the effect of FreezeWorkspace, use the UnfreezeWorkspace Procedure.

Examples

The following example freezes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.FreezeWorkspace ('NEWWORKSPACE');
```

GenerateReplicationSupport Procedure

Creates necessary structures for multimaster replication of Workspace Manager objects, and starts the master activity for the newly created master group.

Syntax

```
DBMS_WM.GenerateReplicationSupport(
   mastersites          IN VARCHAR2,
   groupname            IN VARCHAR2
   [, groupdescription  IN VARCHAR2 DEFAULT 'Replication Group for OWM']);
```

Parameters

Table 80–17 GenerateReplicationSupport Procedure Parameters

Parameter	Description
mastersites	Comma-delimited list of nonwriter site names (database links) to be added to the Workspace Manager replication environment. Do not include the local site (the writer site) in the list.

Table 80–17 GenerateReplicationSupport Procedure Parameters (Cont.)

Parameter	Description
groupname	Name of the master group to be created. This group will appear as a regular replication master group, and it can be managed from all the Oracle replication interfaces, including Oracle Enterprise Manager.
groupdescription	Description of the new master group. The default is Replication Group for OWM.

Usage Notes

To use this procedure, you must understand how replication applies to Workspace Manager objects, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*. You must also understand the major Oracle replication concepts and techniques, which are documented in *Oracle9i Replication* and *Oracle9i Replication Management API Reference*.

You must execute this procedure as the replication administrator user at the writer site.

Before executing this procedure, ensure that the following are true:

- There are no workspaces, savepoints, or version-enabled tables on any of the remote sites specified in the mastersites list
- All the remote sites and the local site have the same version of Workspace Manager installed. You can check the Workspace Manager version number in the WM_INSTALLATION metadata view.
- If there are version-enabled tables on the local site, these tables must exist and must not be version-enabled on each of the remote sites.

This procedure performs the following operations:

- Verifies that the local site and all the sites specified in the mastersites list are running the same version of Workspace Manager.
- Verifies that there are no workspaces, savepoints, or version-enabled tables on any of the remote sites specified in the mastersites list.
- Creates a master group, having the name specified in the groupname parameter, with the local site as the master definition site and the writer site.
- Adds the Workspace Manager metadata tables to this group.
- Disables Workspace Manager operations at all the nonwriter sites (the remote sites specified in the mastersites list).

- If there are any version-enabled tables at the local site, version-enables these tables at each of the remote sites specified in the `mastersites` list and sets them up for replication.
- Starts the master activity for the newly created master group.

To drop replication support for the Workspace Manager environment, use the DropReplicationSupport Procedure.

Examples

The following example generates replication support for the Workspace Manager environment at a hypothetical company.

```
DBMS_WM.GenerateReplicationSupport(
    mastersites       =>  'BACKUP-SITE1.ACME.COM, BACKUP-SITE2.ACME.COM');
    groupname         =>  'OWM-GROUP',
    groupdescription  =>  'OWM Replication group for Acme Corp.');
```

GetConflictWorkspace Function

Returns the name of the workspace on which the session has performed the SetConflictWorkspace Procedure.

Format

```
DBMS_WM.GetConflictWorkspace() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

If the SetConflictWorkspace Procedure has not been executed, the name of the current workspace is returned.

Examples

The following example displays the name of the workspace on which the session has performed the SetConflictWorkspace Procedure.

```
SELECT DBMS_WM.GetConflictWorkspace FROM DUAL;

GETCONFLICTWORKSPACE
--------------------------------------------------------------------------------
```

```
B_focus_2
```

GetDiffVersions Function

Returns the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions Procedure operation.

Format

```
DBMS_WM.GetDiffVersions() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

The returned string is in the format `'(WS1,SP1), (WS2,SP2)'`. This format, including the parentheses, is intended to help you if you later want to use parts of the returned string in a call to the SetDiffVersions Procedure.

Examples

The following example displays the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions Procedure operation.

```
SELECT DBMS_WM.GetDiffVersions FROM DUAL;

GETDIFFVERSIONS
--------------------------------------------------------------------------------
(B_focus_1, LATEST), (B_focus_2, LATEST)
```

GetLockMode Function

Returns the locking mode for the current session, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.

Format

```
DBMS_WM.GetLockMode() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This function returns E, S, C, or NULL.

- For explanations of E (exclusive), S (shared), and C (carry-forward), see the description of the lockmode parameter of the SetLockingON Procedure.
- NULL indicates that locking is not in effect. (Calling the SetLockingOFF Procedure results in this setting.)

For an explanation of Workspace Manager locking, see *Oracle9i Application Developer's Guide - Workspace Manager*. See also the descriptions of the SetLockingON Procedure and SetLockingOFF Procedure.

Examples

The following example displays the locking mode in effect for the session.

```
SELECT DBMS_WM.GetLockMode FROM DUAL;

GETLOCKMODE
--------------------------------------------------------------------------------
C
```

GetMultiWorkspaces Function

Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.

Format

```
DBMS_WM.GetMultiWorkspaces() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This procedure returns the names of workspaces visible in the multiworkspace views, which are described in *Oracle9i Application Developer's Guide - Workspace Manager*.

If no workspaces are visible in the multiworkspace views, NULL is returned. If more than one workspace name is returned, names are separated by a comma (for example: workspace1,workspace2,workspace3).

To make a workspace visible in the multiworkspace views, use the SetMultiWorkspaces Procedure.

Examples

The following example displays the names of workspaces visible in the multiworkspace views.

```
SELECT DBMS_WM.GetMultiWorkspaces FROM DUAL;
```

GetOpContext Function

Returns the context of the current operation for the current session.

Format

```
DBMS_WM.GetOpContext() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This function returns one of the following values:

- DML: The current operation is driven by data manipulation language (DML) initiated by the user.
- MERGE_REMOVE: The current operation was initiated by a MergeWorkspace Procedure call with the remove_workspace parameter set to TRUE or a MergeTable Procedure call with the remove_data parameter set to TRUE.
- MERGE_NOREMOVE: The current operation was initiated by a MergeWorkspace Procedure call with the remove_workspace parameter set to FALSE or a MergeTable Procedure call with the remove_data parameter set to FALSE.

The returned value can be used in user-defined triggers to take appropriate action based on the current operation.

Examples

The following example displays the context of the current operation.

```
SELECT DBMS_WM.GetOpContext FROM DUAL;
```

```
GETOPCONTEXT
--------------------------------------------------------------------------------
DML
```

GetPrivs Function

Returns a comma-delimited list of all privileges that the current user has for the specified workspace.

Format

```
DBMS_WM.GetPrivs(
   workspace  IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 80–18 GetPrivs Function Parameters

Parameter	Description
workspace	Name of the workspace for which to return the list of privileges. The name is case sensitive.

Usage

For information about Workspace Manager privileges, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Examples

The following example displays the privileges that the current user has for the B_focus_2 workspace.

```
SELECT DBMS_WM.GetPrivs ('B_focus_2') FROM DUAL;

DBMS_WM.GETPRIVS('B_FOCUS_2')
--------------------------------------------------------------------------------
ACCESS,MERGE,CREATE,REMOVE,ROLLBACK
```

GetSessionInfo Procedure

Retrieves information about the current workspace and session context.

Format

```
DBMS_WM.GetSessionInfo(
   workspace     OUT VARCHAR2,
   context       OUT VARCHAR2,
   context_type  OUT VARCHAR2);
```

Parameters

Table 80–19 GetSessionInfo Procedure Parameters

Parameter	Description
workspace	Name of the workspace that the current session is in.
context	The context of the current session in the workspace, expressed as one of the following: LATEST, a savepoint name, or an instant (point in time) in 'DD-MON-YYYY HH24:MI:SS' date format. (See the Usage Notes for details.)
context_type	The type of context for the current session in the workspace. Specifically, one of the following values: LATEST (if context is LATEST), SAVEPOINT (if context is a savepoint name), or INSTANT (if context is an instant).

Usage Notes

This procedure is useful if you need to know where a session is (workspace and context) -- for example, after you have performed a combination of GotoWorkspace Procedure, GotoSavepoint Procedure, and GotoDate Procedure operations.

After the procedure successfully executes, the context parameter contains one of the following values:

- LATEST: The session is currently on the LATEST logical savepoint (explained in *Oracle9i Application Developer's Guide - Workspace Manager*), and it can see changes as they are made in the workspace. The context is automatically set to LATEST when the session enters the workspace (using the GotoWorkspace Procedure).
- A savepoint name: The session is currently on a savepoint in the workspace. The session cannot see changes as they are made in the latest version of the workspace, but instead sees a static view of the data as of the savepoint creation time. The session context is set to the savepoint name after a call to the GotoSavepoint Procedure.
- An instant (a point in time): The session is currently on a specific point in time. The session cannot see changes as they are made in the latest version of the

workspace, but instead sees a static view of the data as of the specific time. The session context is set to an instant after a call to the GotoDate Procedure.

For detailed information about the session context, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Examples

The following example retrieves and displays information about the current workspace and context in the session.

```
DECLARE
  current_workspace VARCHAR2(30);
  current_context VARCHAR2(30);
  current_context_type VARCHAR2(30);
BEGIN
  DBMS_WM.GetSessionInfo(current_workspace,
                         current_context,
                         current_context_type);
  DBMS_OUTPUT.PUT_LINE('Session currently in workspace: ' ||current_workspace);
  DBMS_OUTPUT.PUT_LINE('Session context is: ' ||current_context);
  DBMS_OUTPUT.PUT_LINE('Session context is on: ' ||current_context_type);
END;
/
Session currently in workspace: B_focus_2
Session context is: LATEST
Session context is on: LATEST

PL/SQL procedure successfully completed.
```

GetWorkspace Function

Returns the current workspace for the session.

Format

```
DBMS_WM.GetWorkspace() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

None.

Examples

The following example displays the current workspace for the session.

```
SELECT DBMS_WM.GetWorkspace FROM DUAL;

GETWORKSPACE
--------------------------------------------------------------------------------
B_focus_2
```

GotoDate Procedure

Goes to a point at or near the specified date and time in the current workspace.

Syntax

```
DBMS_WM.GotoDate(
   in_date  IN DATE);
```

Parameters

Table 80–20 GotoDate Procedure Parameters

Parameter	Description
in_date	Date and time for the read-only view of the workspace. (See the Usage Notes for details.)

Usage Notes

You are presented a read-only view of the current workspace at or near the specified date and time. The exact time point depends on the history option for tracking changes to data in version-enabled tables, as set by the EnableVersioning Procedure or modified by the SetWoOverwriteOFF Procedure or SetWoOverwriteON Procedure:

- NONE: The read-only view reflects the first savepoint after in_date.
- VIEW_W_OVERWRITE: The read-only view reflects the data values in effect at in_date, except if in_date is between two savepoints and data was changed between the two savepoints. In this case, data that had been changed between the savepoints might be seen as empty or as having a previous value. To ensure the most complete and accurate view of the data, specify the VIEW_WO_OVERWRITE history option when version-enabling a table.

- VIEW_WO_OVERWRITE: The read-only view reflects the data values in effect at in_date.

For an explanation of the history options, see the description of the hist parameter for the EnableVersioning Procedure.

The following example scenario shows the effect of the VIEW_WO_OVERWRITE setting. Assume the following sequence of events:

1. The MANAGER_NAME value in a row is Adams.
2. Savepoint SP1 is created.
3. The MANAGER_NAME value is changed to Baxter.
4. The time point that will be specified as in_date (in step 7) occurs.
5. The MANAGER_NAME value is changed to Chang. (Thus, the value has been changed both before and after in_date since the first savepoint and before the second savepoint.)
6. Savepoint SP2 is created.
7. A GotoDate Procedure operation is executed, specifying the time point in step 4 as in_date.

In the preceding scenario, if the history option in effect is VIEW_WO_OVERWRITE, the MANAGER_NAME value after step 7 is Baxter.

The GotoDate procedure should be executed while users exist in the workspace. There are no explicit privileges associated with this procedure.

Examples

The following example goes to a point at or near midnight at the start of 29-Jun-2001, depending on the history option currently in effect.

```
EXECUTE DBMS_WM.GotoDate ('29-JUN-01');
```

GotoSavepoint Procedure

Goes to the specified savepoint in the current workspace.

Syntax

```
DBMS_WM.GotoSavePoint(
   [savepoint_name  IN VARCHAR2 DEFAULT 'LATEST']);
```

Parameters

Table 80–21 GotoSavepoint Procedure Parameters

Parameter	Description
savepoint_name	Name of the savepoint. The name is case sensitive. If savepoint_name is not specified, the default is LATEST.

Usage Notes

You are presented a read-only view of the workspace at the time of savepoint creation. This procedure is useful for examining the workspace from different savepoints before performing a rollback to a specific savepoint by calling the RollbackToSP Procedure to delete all rows from that savepoint forward.

This operation can be executed while users exist in the workspace. There are no explicit privileges associated with this operation.

If you do not want to roll back to the savepoint, you can call the GotoSavepoint procedure with a null parameter to go to the currently active version in the workspace. (This achieves the same result as calling the GotoWorkspace Procedure and specifying the workspace.)

For more information about savepoints, including the LATEST savepoint, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Examples

The following example goes to the savepoint named Savepoint1.

```
EXECUTE DBMS_WM.GotoSavepoint ('Savepoint1');
```

GotoWorkspace Procedure

Moves the current session to the specified workspace.

Syntax

```
DBMS_WM.GotoWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–22 GotoWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Usage Notes

After a user goes to a workspace, modifications to data can be made there.

To go to the live database, specify workspace as LIVE. Because many operations are prohibited when any users (including you) are in the workspace, it is often convenient to go to the LIVE workspace before performing operations on created workspaces.

An exception is raised if one or more of the following apply:

- workspace does not exist.
- The user does not have ACCESS_WORKSPACE privilege for workspace.
- workspace has been frozen in NO_ACCESS mode (see the FreezeWorkspace Procedure).

Examples

The following example includes the user in the NEWWORKSPACE workspace. The user will begin to work in the latest version in that workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('NEWWORKSPACE');
```

The following example includes the user in the LIVE database workspace. By default, when users connect to a database, they are placed in this workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
```

GrantSystemPriv Procedure

Grants system-level privileges (not restricted to a particular workspace) to users and roles. The grant_option parameter enables the grantee to grant the specified privileges to other users and roles.

Syntax

```
DBMS_WM.GrantSystemPriv(
```

```
   priv_types       IN VARCHAR2,
   grantee          IN VARCHAR2
   [, grant_option  IN VARCHAR2 DEFAULT 'NO']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–23 GrantSystemPriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (*Oracle9i Application Developer's Guide - Workspace Manager* discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, ROLLBACK_ANY_WORKSPACE, and FREEZE_ANY_WORKSPACE.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in priv_types to other users and roles.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

Contrast this procedure with GrantWorkspacePriv Procedure, which grants workspace-level Workspace Manager privileges with keywords that do not contain ANY and which has a workspace parameter.

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For example, assume that user SCOTT has been granted the ACCESS_ANY_WORKSPACE privilege with grant_option as NO, but that the PUBLIC user group has been granted the ACCESS_ANY_WORKSPACE privilege with grant_option as

YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the ACCESS_ANY_WORKSPACE privilege with the grant option.

The WM_ADMIN_ROLE role has all Workspace Manager privileges with the grant option. The WM_ADMIN_ROLE role is automatically given to the DBA role.

The ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other Workspace Manager privileges.

To revoke system-level privileges, use the RevokeSystemPriv Procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You do not have the privilege to grant priv_types.

Examples

The following example enables user Smith to access any workspace in the database, but does not allow Smith to grant the ACCESS_ANY_WORKSPACE privilege to other users.

```
EXECUTE DBMS_WM.GrantSystemPriv ('ACCESS_ANY_WORKSPACE', 'Smith', 'NO');
```

GrantWorkspacePriv Procedure

Grants workspace-level privileges to users and roles. The grant_option parameter enables the grantee to grant the specified privileges to other users and roles.

Syntax

```
DBMS_WM.GrantWorkspacePriv(
   priv_types       IN VARCHAR2,
   workspace        IN VARCHAR2,
   grantee          IN VARCHAR2
   [, grant_option  IN VARCHAR2 DEFAULT 'NO']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–24 GrantWorkspacePriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (*Oracle9i Application Developer's Guide - Workspace Manager* discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, ROLLBACK_WORKSPACE, and FREEZE_WORKSPACE.
workspace	Name of the workspace. The name is case sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in priv_types on the workspace specified in workspace to other users and roles.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

Contrast this procedure with the GrantSystemPriv Procedure, which grants system-level Workspace Manager privileges with keywords in the form *xxx*_ANY_WORKSPACE (ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, and so on).

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For example, assume that user SCOTT has been granted the ACCESS_WORKSPACE privilege with grant_option as NO, but that the PUBLIC user group has been granted the ACCESS_WORKSPACE privilege with grant_option as YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the ACCESS_WORKSPACE privilege with the grant option.

The WM_ADMIN_ROLE role has all Workspace Manager privileges with the grant option. The WM_ADMIN_ROLE role is automatically given to the DBA role.

The ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other Workspace Manager privileges.

To revoke workspace-level privileges, use the RevokeWorkspacePriv Procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You do not have the privilege to grant priv_types.

Examples

The following example enables user Smith to access the NEWWORKSPACE workspace and merge changes in that workspace, and allows Smith to grant the two specified privileges on NEWWORKSPACE to other users.

```
DBMS_WM.GrantWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE', 'NEWWORKSPACE',
'Smith', 'YES');
```

IsWorkspaceOccupied Function

Checks whether or not a workspace has any active sessions.

Syntax

```
DBMS_WM.IsWorkspaceOccupied(
   workspace  IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 80–25 IsWorkspaceOccupied Function Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Usage Notes

This function returns YES if the workspace has any active sessions, and it returns NO if the workspace has no active sessions.

An exception is raised if the LIVE workspace is specified or if the user does not have the privilege to access the workspace.

Examples

The following example checks if any sessions are in the B_focus_2 workspace.

```
SELECT DBMS_WM.IsWorkspaceOccupied('B_focus_2') FROM DUAL;

DBMS_WM.ISWORKSPACEOCCUPIED('B_FOCUS_2')
--------------------------------------------------------------------------------
YES
```

LockRows Procedure

Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.LockRows(
   workspace        IN VARCHAR2,
   table_name       IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, lock_mode     IN VARCHAR2 DEFAULT 'E']);
```

Parameters

Table 80–26 LockRows Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The latest versions of rows visible from the workspace are locked. If a row has not been modified in this workspace, the locked version could be in an ancestor workspace. The name is case sensitive.
table_name	Name of the table in which rows are to be locked. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be locked. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are locked.
lock_mode	Mode with which to set the locks: E (exclusive) or S (shared). The default is E.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. For an explanation of Workspace Manager locking, see *Oracle9i Application Developer's Guide - Workspace Manager*.

This procedure does not affect whether Workspace Manager locking is set on or off (determined by the SetLockingON Procedure and SetLockingOFF Procedure).

To unlock rows, use the UnlockRows Procedure.

Examples

The following example locks rows in the EMPLOYEES table where `last_name = 'Smith'` in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.LockRows ('NEWWORKSPACE', 'employees', 'last_name = ''Smith''');
```

MergeTable Procedure

Applies changes to a table (all rows or as specified in the WHERE clause) in a workspace to its parent workspace.

Syntax

```
DBMS_WM.MergeTable(
   workspace            IN VARCHAR2,
   table_id             IN VARCHAR2
   [, where_clause      IN VARCHAR2 DEFAULT '']
   [, create_savepoint  IN BOOLEAN DEFAULT FALSE]
   [, remove_data       IN BOOLEAN DEFAULT FALSE]
   [, auto_commit       IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–27 MergeTable Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
table_id	Name of the table containing rows to be merged into the parent workspace. The name is not case sensitive.

Table 80–27 MergeTable Procedure Parameters (Cont.)

Parameter	Description
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be merged into the parent workspace. Example: 'department_id = 20'
	Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery.
	If where_clause is not specified, all rows in table_name are merged.
create_savepoint	A Boolean value (TRUE or FALSE).
	TRUE creates an implicit savepoint in the parent workspace before the merge operation. (Implicit and explicit savepoints are described in *Oracle9i Application Developer's Guide - Workspace Manager*.)
	FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.
remove_data	A Boolean value (TRUE or FALSE).
	TRUE removes the data in the table (as specified by where_clause) in the child workspace. This option is permitted only if workspace has no child workspaces (that is, it is a leaf workspace).
	FALSE (the default) does not remove the data in the table in the child workspace.
auto_commit	A Boolean value (TRUE or FALSE).
	TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.
	FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

All data that satisfies the where_clause in the version-enabled table table_name in workspace is applied to the parent workspace of workspace.

Any locks that are held by rows being merged are released.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts

using the <table_name>_CONF view. (Conflict resolution is explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)

A table cannot be merged in the LIVE workspace (because that workspace has no parent workspace).

A table cannot be merged or refreshed if there is an open database transaction affecting the table.

An exception is raised if the user does not have access to table_id, or if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example merges changes to the EMP table (in the USER3 schema) where last_name = 'Smith' in NEWWORKSPACE to its parent workspace.

```
EXECUTE DBMS_WM.MergeTable ('NEWWORKSPACE', 'user3.emp', 'last_name =
''Smith''');
```

MergeWorkspace Procedure

Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.

Syntax

```
DBMS_WM.MergeWorkspace(
   workspace            IN VARCHAR2
   [, create_savepoint  IN BOOLEAN DEFAULT FALSE]
   [, remove_workspace  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit       IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–28 MergeWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Table 80–28 MergeWorkspace Procedure Parameters (Cont.)

Parameter	Description
create_savepoint	A Boolean value (TRUE or FALSE). TRUE creates an implicit savepoint in the parent workspace before the merge operation. (Implicit and explicit savepoints are described in *Oracle9i Application Developer's Guide - Workspace Manager*.) FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.
remove_workspace	A Boolean value (TRUE or FALSE). TRUE removes workspace after the merge operation. FALSE (the default) does not remove workspace after the merge operation; the workspace continues to exist.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

All data in all version-enabled tables in workspace is merged to the parent workspace of workspace, and workspace is removed if remove_workspace is TRUE.

While this procedure is executing, the current workspace is frozen in NO_ACCESS mode and the parent workspace is frozen in READ_ONLY mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)

If the remove_workspace parameter value is TRUE, the workspace to be merged must be a leaf workspace, that is, a workspace with no descendant workspaces. (For

an explanation of workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.)

An exception is raised if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example merges changes in NEWWORKSPACE to its parent workspace and removes (by default) NEWWORKSPACE.

```
EXECUTE DBMS_WM.MergeWorkspace ('NEWWORKSPACE');
```

RecoverAllMigratingTables Procedure

Attempts to complete the migration process on all tables that were left in an inconsistent state after the Workspace Manager migration procedure failed.

Syntax

```
DBMS_WM.RecoverAllMigratingTables(
   [, ignore_last_error IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–29 RecoverAllMigratingTables Procedure Parameters

Parameter	Description
ignore_ last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the migration process. Information about the last error is stored in the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views, which are described in *Oracle9i Application Developer's Guide - Workspace Manager*. (See the Usage Notes for more information.) FALSE (the default) does not ignore the last error, if any, that occurred during the migration process.

Usage Notes

If an error occurs while you are upgrading to the current Workspace Manager release, one or more version-enabled tables can be left in an inconsistent state. (For information about upgrading to the current release, see *Oracle9i Application Developer's Guide - Workspace Manager*.) If the upgrade procedure fails, you should try to fix the cause of the error (examine the USER_WM_VT_ERRORS or ALL_WM_VT_

ERRORS metadata view to see the SQL statement and error message), and then call the RecoverMigratingTable Procedure (for a single table) or RecoverAllMigratingTables Procedure (for all tables) with the default ignore_last_error parameter value of FALSE, to try to complete the upgrade process.

However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the RecoverMigratingTable Procedure or RecoverAllMigratingTables Procedure with the ignore_last_error parameter value of TRUE. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

Examples

The following example attempts to recover all version-enabled tables that were left in an inconsistent state when the upgrade procedure failed.

```
EXECUTE DBMS_WM.RecoverAllMigratingTables;
```

The following example attempts to recover all version-enabled tables that were left in an inconsistent state when the upgrade procedure failed, and it ignores the last error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverAllMigratingTables(TRUE);
```

RecoverMigratingTable Procedure

Attempts to complete the migration process on a table that was left in an inconsistent state after the Workspace Manager migration procedure failed.

Syntax

```
DBMS_WM.RecoverMigratingTable(
   table_name           IN VARCHAR2
   [, ignore_last_error IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–30 RecoverMigratingTable Procedure Parameters

Parameter	Description
table_name	Name of the version-enabled table to be recovered from the migration error. The name is not case sensitive.

Table 80–30 RecoverMigratingTable Procedure Parameters (Cont.)

Parameter	Description
ignore_ last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the migration process. Information about the last error is stored in the USER_WM_VT_ERRORS and ALL_WM_VT_ERRORS metadata views, which are described in *Oracle9i Application Developer's Guide - Workspace Manager*. (See the Usage Notes for more information.) FALSE (the default) does not ignore the last error, if any, that occurred during the migration process.

Usage Notes

If an error occurs while you are upgrading to the current Workspace Manager release, one or more version-enabled tables can be left in an inconsistent state. (For information about upgrading to the current release, see *Oracle9i Application Developer's Guide - Workspace Manager*.) If the upgrade procedure fails, you should try to fix the cause of the error (examine the USER_WM_VT_ERRORS or ALL_WM_VT_ERRORS metadata view to see the SQL statement and error message), and then call the RecoverMigratingTable Procedure (for a single table) or RecoverAllMigratingTables Procedure (for all tables) with the default ignore_last_error parameter value of FALSE, to try to complete the upgrade process.

However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the RecoverMigratingTable Procedure or RecoverAllMigratingTables Procedure with the ignore_last_error parameter value of TRUE. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if table_name does not exist or is not version-enabled.

Examples

The following example attempts to recover the COLA_MARKETING_BUDGET table from the error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverMigratingTable('COLA_MARKETING_BUDGET');
```

The following example attempts to recover the COLA_MARKETING_BUDGET table and ignores the last error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverMigratingTable('COLA_MARKETING_BUDGET', TRUE);
```

RefreshTable Procedure

Applies to a workspace all changes made to a table (all rows or as specified in the WHERE clause) in its parent workspace.

Syntax

```
DBMS_WM.RefreshTable(
   workspace        IN VARCHAR2,
   table_id         IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–31 RefreshTable Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
table_id	Name of the table containing the rows to be refreshed using values from the parent workspace. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be refreshed from the parent workspace. Example: 'department_id = 20'
	Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery.
	If where_clause is not specified, all rows in table_name are refreshed.
auto_commit	A Boolean value (TRUE or FALSE).
	TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.
	FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

This procedure applies to workspace all changes in rows that satisfy the where_clause in the version-enabled table table_id in the parent workspace since the time when workspace was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)

A table cannot be refreshed in the LIVE workspace (because that workspace has no parent workspace).

A table cannot be merged or refreshed if there is an open database transaction affecting the table.

An exception is raised if the user does not have access to table_id, or if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example refreshes NEWWORKSPACE by applying changes made to the EMPLOYEES table where last_name = 'Smith' in its parent workspace.

```
EXECUTE DBMS_WM.RefreshTable ('NEWWORKSPACE', 'employees', 'last_name =
''Smith''');
```

RefreshWorkspace Procedure

Applies to a workspace all changes made in its parent workspace.

Syntax

```
DBMS_WM.RefreshWorkspace(
   workspace      IN VARCHAR2
   [, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–32 RefreshWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Table 80–32 RefreshWorkspace Procedure Parameters (Cont.)

Parameter	Description
auto_ commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

This procedure applies to workspace all changes made to version-enabled tables in the parent workspace since the time when workspace was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)

The specified workspace and the parent workspace are frozen in READ_ONLY mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

The LIVE workspace cannot be refreshed (because it has no parent workspace).

An exception is raised if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example refreshes NEWWORKSPACE by applying changes made in its parent workspace.

```
EXECUTE DBMS_WM.RefreshWorkspace ('NEWWORKSPACE');
```

RelocateWriterSite Procedure

Makes one of the nonwriter sites the new writer site in a Workspace Manager replication environment. (The old writer site becomes one of the nonwriter sites.)

Syntax

```
DBMS_WM.RelocateWriterSite(
   newwritersite           IN VARCHAR2,
   oldwritersiteavailable  IN BOOLEAN);
```

Parameters

Table 80–33 RelocateWriterSite Procedure Parameters

Parameter	Description
`newwritersite`	Name of a current nonwriter site names (database link) to be made the new writer site in the Workspace Manager replication environment.
`oldwritersiteavailable`	A Boolean value (TRUE or FALSE). TRUE causes the old writer site to be updated to reflect the fact that the writer site has changed. FALSE causes the old writer site not to be updated to reflect the fact that the writer site has changed. In this case, you must use the SynchronizeSite Procedure when the old writer site becomes available.

Usage Notes

To use this procedure, you must understand how replication applies to Workspace Manager objects, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*. You must also understand the major Oracle replication concepts and techniques, which are documented in *Oracle9i Replication* and *Oracle9i Replication Management API Reference*.

You must execute this procedure as the replication administrator user. You can execute it at any master site.

You should specify the `oldwritersiteavailable` parameter as TRUE if the old writer site is currently available. If you specify the `oldwritersiteavailable` parameter as FALSE, you must execute the SynchronizeSite Procedure after the old writer site becomes available, to bring that site up to date.

This procedure performs the following operations:

- If `oldwritersiteavailable` is TRUE, disables workspace operations and DML and DDL operations for all version-enabled tables on the old writer site.
- Enables workspace operations and DML and DDL operations for all version-enabled tables on the new writer site.

- Invokes replication API procedures to relocate the master definition site to `newwritersite` for the main master group and for the master groups for all the version-enabled tables.

Examples

The following example relocates the writer site for the Workspace Manager environment to `BACKUP-SITE1` at a hypothetical company.

```
DBMS_WM.RelocateWriterSite(
    newwritersite           =>  'BACKUP-SITE1.ACME.COM');
    oldwritersiteavailable  =>  TRUE);
```

RemoveWorkspace Procedure

Discards all row versions associated with a workspace and deletes the workspace.

Syntax

```
DBMS_WM.RemoveWorkspace(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–34 RemoveWorkspace Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.
`auto_commit`	A Boolean value (`TRUE` or `FALSE`). `TRUE` (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. `FALSE` causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

The RemoveWorkspace operation can only be performed on leaf workspaces (the bottom-most workspaces in a branch in the hierarchy). For an explanation of

database workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.

There must be no other users in the workspace being removed.

An exception is raised if the user does not have the REMOVE_WORKSPACE privilege for workspace or the REMOVE_ANY_WORKSPACE privilege.

Examples

The following example removes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.RemoveWorkspace('NEWWORKSPACE');
```

RemoveWorkspaceTree Procedure

Discards all row versions associated with a workspace and its descendant workspaces, and deletes the affected workspaces.

Syntax

```
DBMS_WM.RemoveWorkspaceTree(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–35 RemoveWorkspaceTree Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

The RemoveWorkspaceTree operation should be used with extreme caution, because it removes support structures and rolls back changes in a workspace and all

its descendants down to the leaf workspace or workspaces. For an explanation of database workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.

There must be no other users in `workspace` or any of its descendant workspaces.

An exception is raised if the user does not have the `REMOVE_WORKSPACE` privilege for `workspace` or any of its descendant workspaces.

Examples

The following example removes the `NEWWORKSPACE` workspace and all its descendant workspaces.

```
EXECUTE DBMS_WM.RemoveWorkspaceTree('NEWWORKSPACE');
```

ResolveConflicts Procedure

Resolves conflicts between workspaces.

Syntax

```
DBMS_WM.ResolveConflicts(
   workspace     IN VARCHAR2,
   table_name    IN VARCHAR2,
   where_clause  IN VARCHAR2,
   keep          IN VARCHAR2);
```

Parameters

Table 80–36 ResolveConflicts Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace to check for conflicts with other workspaces. The name is case sensitive.
`table_name`	Name of the table to check for conflicts. The name is not case sensitive.
`where_clause`	The `WHERE` clause (excluding the `WHERE` keyword) identifying the rows to be refreshed from the parent workspace. Example: `'department_id = 20'` Only primary key columns can be specified in the `WHERE` clause. The `WHERE` clause cannot contain a subquery.

Table 80–36 ResolveConflicts Procedure Parameters (Cont.)

Parameter	Description
keep	Workspace in favor of which to resolve conflicts: PARENT, CHILD, or BASE. PARENT causes the parent workspace rows to be copied to the child workspace. CHILD does not cause the child workspace rows to be copied immediately to the parent workspace. However, the conflict is considered resolved, and the child workspace rows are copied to the parent workspace when the child workspace is merged. BASE causes the base rows to be copied to the child workspace but not to the parent workspace. However, the conflict is considered resolved; and when the child workspace is merged, the base rows are copied to the parent workspace. Note that BASE is ignored for insert-insert conflicts where a base row does not exist; in this case the keep parameter value must be PARENT or CHILD.

Usage Notes

This procedure checks the condition identified by `table_name` and `where_clause`, and it finds any conflicts between row values in `workspace` and its parent workspace. This procedure resolves conflicts by using the row values in the parent or child workspace, as specified in the `keep` parameter; however, the conflict resolution is not actually merged until you commit the transaction (standard database commit operation) and call the CommitResolve Procedure to end the conflict resolution session. (For more information about conflict resolution, including an overall view of the process, see *Oracle9i Application Developer's Guide - Workspace Manager*.)

For example, assume that for Department 20 (`DEPARTMENT_ID = 20`), the `MANAGER_NAME` in the `LIVE` and `Workspace1` workspaces is `Tom`. Then, the following operations occur:

1. The `manager_name` for Department 20 is changed in the `LIVE` database workspace from `Tom` to `Mary`.
2. The change is committed (a standard database commit operation).
3. The `manager_name` for Department 20 is changed in `Workspace1` from `Tom` to `Franco`.
4. The MergeWorkspace Procedure is called to merge `Workspace1` changes to the `LIVE` workspace.

At this point, however, a conflict exists with respect to MANAGER_NAME for Department 20 in Workspace1 (Franco, which conflicts with Mary in the LIVE workspace), and therefore the call to MergeWorkspace Procedure does not succeed.

5. The ResolveConflicts procedure is called with the following parameters: ('Workspace1', 'department', 'department_id = 20', 'child').

   After the MergeWorkspace Procedure operation in step 7, the MANAGER_NAME value will be Franco in both the Workspace1 and LIVE workspaces.

6. The change is committed (a standard database commit operation).

7. The MergeWorkspace Procedure is called to merge Workspace1 changes to the LIVE workspace.

For more information about conflict resolution, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Examples

The following example resolves conflicts involving rows in the DEPARTMENT table in Workspace1 where DEPARTMENT_ID is 20, and uses the values in the child workspace to resolve all such conflicts. It then merges the results of the conflict resolution by first committing the transaction (standard commit) and then calling the MergeWorkspace Procedure.

```
EXECUTE DBMS_WM.BeginResolve ('Workspace1');
EXECUTE  DBMS_WM.ResolveConflicts ('Workspace1', 'department', 'department_id =
20', 'child');
COMMIT;
EXECUTE DBMS_WM.CommitResolve ('Workspace1');
```

RevokeSystemPriv Procedure

Revokes (removes) system-level privileges from users and roles.

Syntax

```
DBMS_WM.RevokeSystemPriv(
   priv_types      IN VARCHAR2,
   grantee         IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–37 RevokeSystemPriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (*Oracle9i Application Developer's Guide - Workspace Manager* discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

Contrast this procedure with the RevokeWorkspacePriv Procedure, which revokes workspace-level Workspace Manager privileges with keywords in the form *xxx*_WORKSPACE (ACCESS_WORKSPACE, MERGE_WORKSPACE, and so on).

To grant system-level privileges, use the GrantSystemPriv Procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You were not the grantor of priv_types to grantee.

Examples

The following example disallows user Smith from accessing workspaces and merging changes in workspaces.

```
EXECUTE DBMS_WM.RevokeSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE',
'Smith');
```

RevokeWorkspacePriv Procedure

Revokes (removes) workspace-level privileges from users and roles for a specified workspace.

Syntax

```
DBMS_WM.RevokeWorkspacePriv(
   priv_types      IN VARCHAR2,
   workspace       IN VARCHAR2,
   grantee         IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–38 RevokeWorkspacePriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (*Oracle9i Application Developer's Guide - Workspace Manager* discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.
workspace	Name of the workspace. The name is case sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

Contrast this procedure with the RevokeSystemPriv Procedure, which revokes system-level Workspace Manager privileges with keywords in the form *xxx*_ANY_WORKSPACE (ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, and so on).

To grant workspace-level privileges, use the GrantWorkspacePriv Procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You were not the grantor of priv_types to grantee.

Examples

The following example disallows user Smith from accessing the NEWWORKSPACE workspace and merging changes in that workspace.

```
EXECUTE DBMS_WM.RevokeWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE',
'NEWWORKSPACE', 'Smith');
```

RollbackDDL Procedure

Rolls back (cancels) DDL (data definition language) changes made during a DDL session for a specified table, and ends the DDL session.

Syntax

```
DBMS_WM.RollbackDDL(
   table_name  IN VARCHAR2);
```

Parameters

Table 80–39 RollbackDDL Procedure Parameters

Parameter	Description
table_name	Name of the version-enabled table. The name is not case sensitive.

Usage Notes

This procedure rolls back (cancels) changes that were made to a version-enabled table and to any indexes and triggers based on the version-enabled table during a DDL session. It also deletes the special *<table-name>_LTS* table that had been created by the BeginDDL Procedure.

For detailed information about performing DDL operations related to version-enabled tables and about DDL operations on version-enabled tables in an Oracle replication environment, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- table_name does not exist or is not version-enabled.
- An open DDL session does not exist for table_name. (That is, the BeginDDL Procedure has not been called specifying this table, or the CommitDDL Procedure or RollbackDDL Procedure was already called specifying this table.)

Examples

The following example begins a DDL session, adds a column named COMMENTS to the COLA_MARKETING_BUDGET table by using the special table named COLA_MARKETING_BUDGET_LTS, and ends the DDL session by canceling the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
EXECUTE DBMS_WM.RollbackDDL('COLA_MARKETING_BUDGET');
```

RollbackResolve Procedure

Quits a conflict resolution session and discards all changes in the workspace since the BeginResolve Procedure was executed.

Syntax

```
DBMS_WM.RollbackResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–40 RollbackResolve Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Usage Notes

This procedure quits the current conflict resolution session (started by the BeginResolve Procedure), and discards all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with the CommitResolve Procedure, which saves all changes.

While the conflict resolution session is being rolled back, the workspace is frozen in 1WRITER mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

For more information about conflict resolution, see *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The procedure was called by a user that does not have the `WM_ADMIN_ROLE` role or that did not execute the BeginResolve Procedure on `workspace`.

Examples

The following example quits the conflict resolution session in `Workspace1` and discards all changes.

```
EXECUTE  DBMS_WM.RollbackResolve ('Workspace1');
```

RollbackTable Procedure

Discards all changes made in the workspace to a specified table (all rows or as specified in the `WHERE` clause).

Syntax

```
DBMS_WM.RollbackTable(
   workspace        IN VARCHAR2,
   table_id         IN VARCHAR2,
   [, sp_name       IN VARCHAR2 DEFAULT '']
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, remove_locks  IN BOOLEAN DEFAULT TRUE]
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–41 RollbackTable Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.
`table_id`	Name of the table containing rows to be discarded. The name is not case sensitive.
`sp_name`	Name of the savepoint to which to roll back. The name is case sensitive. The default is to discard all changes (that is, ignore any savepoints).

Table 80–41 RollbackTable Procedure Parameters (Cont.)

Parameter	Description
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be discarded. Example: 'department_id = 20'
	Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery.
	If where_clause is not specified, all rows that meet the criteria of the other parameters are discarded.
remove_locks	A Boolean value (TRUE or FALSE).
	TRUE (the default) releases those locks on rows in the parent workspace that satisfy the condition in where_clause and that were not versioned in the child workspace. This option has no effect if a savepoint is specified (sp_name parameter).
	FALSE does not release any locks in the parent workspace.
auto_commit	A Boolean value (TRUE or FALSE).
	TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.
	FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

You cannot roll back to a savepoint if any implicit savepoints have been created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created.

An exception is raised if one or more of the following apply:

- workspace does not exist.
- You do not have the privilege to roll back workspace or any affected table.
- A database transaction affecting table_id is open in workspace.

Examples

The following example rolls back all changes made to the EMP table (in the USER3 schema) in the NEWWORKSPACE workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackTable ('NEWWORKSPACE', 'user3.emp');
```

RollbackToSP Procedure

Discards all data changes made in the workspace to version-enabled tables since the specified savepoint.

Syntax

```
DBMS_WM.RollbackToSP(
   workspace       IN VARCHAR2,
   savepoint_name  IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–42 RollbackToSP Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
savepoint_name	Name of the savepoint to which to roll back changes. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

While this procedure is executing, the workspace is frozen in NO_ACCESS mode.

Contrast this procedure with the RollbackWorkspace Procedure, which rolls back all changes made since the creation of the workspace.

You cannot roll back to a savepoint if any implicit savepoints have been created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created.

An exception is raised if one or more of the following apply:

- workspace does not exist.

- savepoint_name does not exist.
- One or more implicit savepoints have been created in workspace after savepoint_name, and the descendant workspaces that caused the implicit savepoints to be created still exist.
- You do not have the privilege to roll back workspace or any affected table.
- Any sessions are in workspace.

Examples

The following example rolls back any changes made in the NEWWORKSPACE workspace to all tables since the creation of Savepoint1.

```
EXECUTE DBMS_WM.RollbackToSP ('NEWWORKSPACE', 'Savepoint1');
```

RollbackWorkspace Procedure

Discards all data changes made in the workspace to version-enabled tables.

Syntax

```
DBMS_WM.RollbackWorkspace(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 80–43 RollbackWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see *Oracle9i Application Developer's Guide - Workspace Manager*.

Usage Notes

Only leaf workspaces can be rolled back. That is, a workspace cannot be rolled back if it has any descendant workspaces. (For an explanation of workspace hierarchy, see *Oracle9i Application Developer's Guide - Workspace Manager*.)

Contrast this procedure with the RollbackToSP Procedure, which rolls back changes to a specified savepoint.

Like the RemoveWorkspace Procedure, RollbackWorkspace deletes the data in the workspace; however, unlike the RemoveWorkspace Procedure, RollbackWorkspace does not delete the Workspace Manager workspace structure.

While this procedure is executing, the specified workspace is frozen in NO_ACCESS mode, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*.

An exception is raised if one or more of the following apply:

- `workspace` has any descendant workspaces.
- `workspace` does not exist.
- You do not have the privilege to roll back `workspace` or any affected table.
- Any sessions are in `workspace`.

Examples

The following example rolls back any changes made in the NEWWORKSPACE workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackWorkspace ('NEWWORKSPACE');
```

SetConflictWorkspace Procedure

Determines whether or not conflicts exist between a workspace and its parent.

Syntax

```
DBMS_WM.SetConflictWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–44 SetConflictWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.

Usage Notes

This procedure checks for any conflicts between `workspace` and its parent workspace, and it modifies the content of the <table_name>_CONF views (explained in *Oracle9i Application Developer's Guide - Workspace Manager*) as needed.

A `SELECT` operation from the <table_name>_CONF views for all tables modified in a workspace displays all rows in the workspace that are in conflict with the parent workspace. (To obtain a list of tables that have conflicts for the current conflict workspace setting, use the SQL statement `SELECT * FROM ALL_WM_VERSIONED_TABLES WHERE conflict = 'YES';`. The SQL statement `SELECT * FROM <table_name>_CONF` displays conflicts for <table_name> between the current workspace and its parent workspace.)

Any conflicts must be resolved before a workspace can be merged or refreshed. To resolve a conflict, you must use the ResolveConflicts Procedure (and then merge the result of the resolution by using the MergeWorkspace Procedure).

Examples

The following example checks for any conflicts between `B_focus_2` and its parent workspace, and modifies the contents of the <table_name>_CONF views as needed.

```
EXECUTE DBMS_WM.SetConflictWorkspace ('B_focus_2');
```

SetDiffVersions Procedure

Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It modifies the contents of the differences views that describe these differences.

Syntax

```
DBMS_WM.SetDiffVersions(
   workspace1  IN VARCHAR2,
   workspace2  IN VARCHAR2);
```

or

```
DBMS_WM.SetDiffVersions(
   workspace1  IN VARCHAR2,
   savepoint1  IN VARCHAR2,
   workspace2  IN VARCHAR2,
   savepoint2  IN VARCHAR2);
```

Parameters

Table 80–45 SetDiffVersions Procedure Parameters

Parameter	Description
workspace1	Name of the first workspace to be checked for differences in version-enabled tables. The name is case sensitive.
savepoint1	Name of the savepoint in workspace1 for which values are to be checked. The name is case sensitive. If savepoint1 and savepoint2 are not specified, the rows in version-enabled tables for the LATEST savepoint in each workspace are checked. (The LATEST savepoint is explained in *Oracle9i Application Developer's Guide - Workspace Manager*.)
workspace2	Name of the second workspace to be checked for differences in version-enabled tables. The name is case sensitive.
savepoint2	Name of the savepoint in workspace2 for which values are to be checked. The name is case sensitive.

Usage Notes

This procedure modifies the contents of the differences views (xxx_DIFF), which are described in *Oracle9i Application Developer's Guide - Workspace Manager*. Each call to the procedure populates one or more sets of three rows, each set consisting of:

- Values for the common ancestor
- Values for workspace1 (savepoint1 or LATEST savepoint values)
- Values for workspace2 (savepoint2 or LATEST savepoint values)

You can then select rows from the appropriate xxx_DIFF view or views to check comparable table values in the two savepoints and their common ancestor. The common ancestor (or *base*) is identified as DiffBase in xxx_DIFF view rows.

Examples

The following example checks the differences in version-enabled tables for the B_focus_1 and B_focus_2 workspaces. (The output has been reformatted for readability.)

```
SQL> -- Add rows to difference view: COLA_MARKETING_BUDGET_DIFF
SQL> EXECUTE DBMS_WM.SetDiffVersions ('B_focus_1', 'B_focus_2');

SQL> -- View the rows that were just added.
SQL> SELECT * from COLA_MARKETING_BUDGET_DIFF;

PRODUCT_ID  PRODUCT_NAME  MANAGER  BUDGET  WM_DIFFVER          WMCODE
----------  ------------  -------  ------  -----------         --------
       1        cola_a    Alvarez   2      DiffBase            NC
       1        cola_a    Alvarez   1.5    B_focus_1, LATEST   U
       1        cola_a    Alvarez   2      B_focus_2, LATEST   NC
       2        cola_b    Burton    2      DiffBase            NC
       2        cola_b    Beasley   3      B_focus_1, LATEST   U
       2        cola_b    Burton    2.5    B_focus_2, LATEST   U
       3        cola_c    Chen      1.5    DiffBase            NC
       3        cola_c    Chen      1      B_focus_1, LATEST   U
       3        cola_c    Chen      1.5    B_focus_2, LATEST   NC
       4        cola_d    Davis     3.5    DiffBase            NC
       4        cola_d    Davis     3      B_focus_1, LATEST   U
       4        cola_d    Davis     2.5    B_focus_2, LATEST   U

12 rows selected.
```

Oracle9i Application Developer's Guide - Workspace Manager explains how to interpret and use the information in the differences (xxx_DIFF) views.

SetLockingOFF Procedure

Disables Workspace Manager locking for the current session.

Syntax

```
DBMS_WM.SetLockingOFF();
```

Parameters

None.

Usage Notes

This procedure turns off Workspace Manager locking that had been set on by the SetLockingON Procedure. Existing locks applied by this session remain locked. All new changes by this session are not locked.

Examples

The following example sets locking off for the session.

```
EXECUTE DBMS_WM.SetLockingOFF;
```

SetLockingON Procedure

Enables Workspace Manager locking for the current session.

Syntax

```
DBMS_WM.SetLockingON(
   lockmode  IN VARCHAR2);
```

Parameters

Table 80–46 SetLockingON Procedure Parameters

Parameter	Description
lockmode	Locking mode. Must be E, S, or C.
	E (exclusive) mode locks the rows in the previous version and the corresponding rows in the current version; no other users in the workspace for either version can change any values.
	S (shared) mode locks the rows in the previous version and the corresponding rows in the current version; however, other users in the workspace for the current version (but no users in the workspace for the previous version) can change values in these rows.
	C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the previous version. (If a row is not locked in the previous version, its corresponding row in the current version is not locked.)

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace

is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

Exclusive locking prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

Locking is enabled at the user session level, and the locking mode stays in effect until any of the following occurs:

- The session goes to another workspace or connects to the database, in which case the locking mode is set to C (carry-forward) unless another locking mode has been specified using the SetWorkspaceLockModeON Procedure.
- The session executes the SetLockingOFF Procedure.

The locks remain in effect for the duration of the workspace, unless unlocked by the UnlockRows Procedure. (Existing locks are not affected by the SetLockingOFF Procedure.)

There are no specific privileges associated with locking. Any session that can go to a workspace can set locking on.

Examples

The following example sets exclusive locking on for the session.

```
EXECUTE DBMS_WM.SetLockingON ('E');
```

All rows locked by this user remain locked until the workspace is merged or rolled back.

SetMultiWorkspaces Procedure

Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.

Syntax

```
DBMS_WM.SetMultiWorkspaces(
   workspaces  IN VARCHAR2);
```

Parameters

Table 80–47 SetMultiWorkspaces Procedure Parameters

Parameter	Description
workspaces	The workspace or workspaces for which information is to be added to the multiworkspace views (described in *Oracle9i Application Developer's Guide - Workspace Manager*). The workspace names are case sensitive. To specify more than one workspace (but no more than eight), use a comma to separate workspace names. For example: 'workspace1,workspace2'

Usage Notes

This procedure adds rows to the multiworkspace views (xxx_MW). See *Oracle9i Application Developer's Guide - Workspace Manager* for information about the contents and uses of these views.

To see the names of workspaces visible in the multiworkspace views, use the GetMultiWorkspaces Function.

An exception is raised if one or more of the following apply:

- The user does not have the privilege to go to one or more of the workspaces named in workspaces.
- A workspace named in workspaces is not valid.

Examples

The following example adds information to the multiworkspace views for version-enabled tables in the B_focus_1 workspace.

```
SQL> EXECUTE DBMS_WM.SetMultiWorkspaces ('B_focus_1');
```

SetWoOverwriteOFF Procedure

Disables the VIEW_WO_OVERWRITE history option that had been enabled by the EnableVersioning Procedure or SetWoOverwriteON Procedure, changing the option to VIEW_W_OVERWRITE (*with overwrite*).

Syntax

```
DBMS_WM.SetWoOverwriteOFF();
```

Parameters

None.

Usage Notes

This procedure affects the recording of history information in the views named <table_name>_HIST by changing the VIEW_WO_OVERWRITE option to VIEW_W_OVERWRITE. That is, from this point forward, the views show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes.

This procedure affects only tables that were version-enabled with the hist parameter set to VIEW_WO_OVERWRITE in the call to the EnableVersioning Procedure.

The <table_name>_HIST views are described in *Oracle9i Application Developer's Guide - Workspace Manager*. The VIEW_WO_OVERWRITE and VIEW_W_OVERWRITE options are further described in the description of the EnableVersioning Procedure.

The history option affects the behavior of the GotoDate Procedure. See the Usage Notes for that procedure.

The result of the SetWoOverwriteOFF procedure remains in effect only for the duration of the current session. To reverse the effect of this procedure, use the SetWoOverwriteON Procedure.

Examples

The following example disables the VIEW_WO_OVERWRITE history option.

```
EXECUTE DBMS_WM.SetWoOverwriteOFF;
```

SetWoOverwriteON Procedure

Enables the VIEW_WO_OVERWRITE history option that had been disabled by the SetWoOverwriteOFF Procedure.

Syntax

```
DBMS_WM.SetWoOverwriteON();
```

Parameters

None.

Usage Notes

This procedure affects the recording of history information in the views named <table_name>_HIST by changing the VIEW_W_OVERWRITE option to VIEW_WO_OVERWRITE (*without overwrite*). That is, from this point forward, the views show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.

This procedure affects only tables that were affected by a previous call to the SetWoOverwriteOFF Procedure.

The <table_name>_HIST views are described in *Oracle9i Application Developer's Guide - Workspace Manager*. The VIEW_WO_OVERWRITE and VIEW_W_OVERWRITE options are further described in the description of the EnableVersioning Procedure.

The VIEW_WO_OVERWRITE history option can be overridden when a workspace is compressed by specifying the compress_view_wo_overwrite parameter as TRUE with the CompressWorkspace Procedure or CompressWorkspaceTree Procedure.

The history option affects the behavior of the GotoDate Procedure. See the Usage Notes for that procedure.

To reverse the effect of this procedure, use the SetWoOverwriteOFF Procedure.

Examples

The following example enables the VIEW_WO_OVERWRITE history option.

```
EXECUTE DBMS_WM.SetWoOverwriteON;
```

SetWorkspaceLockModeOFF Procedure

Disables Workspace Manager locking for the specified workspace.

Syntax

```
DBMS_WM.SetWorkspaceLockModeOFF(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–48 SetWorkspaceLockModeOFF Procedure Parameters

Parameter	Description
workspace	Name of the workspace for which to set the locking mode off. The name is case sensitive.

Usage Notes

This procedure turns off Workspace Manager locking that had been set on by the SetWorkspaceLockModeON Procedure. Existing locks applied by this session remain locked. All new changes by this session or a subsequent session are not locked, unless the session turns locking on by executing the SetLockingON Procedure.

An exception is raised if any of the following occurs:

- The user does not have the WM_ADMIN_ROLE role or is not the owner of workspace.
- There are any open database transactions in workspace.
- workspace is a continually refreshed workspace (see the description of the isrefreshed parameter of the CreateWorkspace Procedure).

Examples

The following example sets locking off for the workspace named NEWWORKSPACE.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeOFF('NEWWORKSPACE');
```

SetWorkspaceLockModeON Procedure

Enables Workspace Manager locking for the specified workspace.

Syntax

```
DBMS_WM.SetWorkspaceLockModeON(
   workspace    IN VARCHAR2,
   lockmode     IN VARCHAR2
   [, override  IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 80–49 SetWorkspaceLockModeON Procedure Parameters

Parameter	Description
workspace	Name of the workspace for which to enable Workspace Manager locking. The name is case sensitive.
lockmode	Default locking mode for row-level locking. Must be E, S, or C. E (exclusive) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; no other users in either workspace can change any values. S (shared) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; however, other users in the current workspace (but no users in the parent workspace) can change values in these rows. C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the parent workspace. (If a row is not locked in the parent workspace, its corresponding row in the child workspace is not locked.)
override	A Boolean value (TRUE or FALSE) TRUE allows a session in the workspace to change the lockmode value by using the SetLockingON Procedure and SetLockingOFF Procedure. FALSE (the default) prevents a session in the workspace from changing the lockmode value.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

Exclusive locking prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

If the override parameter value is TRUE, locking can also be enabled and disabled at the user session level with the SetLockingON Procedure and SetLockingOFF Procedure, respectively.

All new changes by this session or a subsequent session are locked, unless the session turns locking off by executing the SetLockingOFF Procedure.

An exception is raised if any of the following occurs:

- The user does not have the WM_ADMIN_ROLE role or is not the owner of workspace.
- There are any open database transactions in workspace.
- workspace is a continually refreshed workspace (see the description of the isrefreshed parameter of the CreateWorkspace Procedure).

Examples

The following example sets exclusive locking on for the workspace named NEWWORKSPACE.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeON ('NEWWORKSPACE', 'E');
```

All locked rows remain locked until the workspace is merged or rolled back.

SynchronizeSite Procedure

Brings the local site (the old writer site) up to date in the Workspace Manager replication environment after the writer site was moved using the RelocateWriterSite Procedure.

Syntax

```
DBMS_WM.SynchronizeSite(
   newwritersite  IN VARCHAR2);
```

Parameters

Table 80–50 SynchronizeSite Procedure Parameters

Parameter	Description
newwritersite	Name of the new writer site (database link) with which the local site needs to be brought up to date.

Usage Notes

To use this procedure, you must understand how replication applies to Workspace Manager objects, as explained in *Oracle9i Application Developer's Guide - Workspace Manager*. You must also understand the major Oracle replication concepts and techniques, which are documented in *Oracle9i Replication* and *Oracle9i Replication Management API Reference*.

You must execute this procedure as the replication administrator user.

You must execute this procedure on the old writer site if you specified the `oldwritersiteavailable` parameter as `FALSE` when you executed the RelocateWriterSite Procedure.

Examples

The following example brings the local system up to date with the new writer site (`BACKUP-SITE1.ACME.COM`) in the Workspace Manager replication environment.

```
DBMS_WM.SynchronizeSite('BACKUP-SITE1.ACME.COM');
```

UnfreezeWorkspace Procedure

Enables access and changes to a workspace, reversing the effect of the FreezeWorkspace Procedure.

Syntax

```
DBMS_WM.UnfreezeWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 80–51 UnfreezeWorkspace Procedure Parameters

Parameter	Description
`workspace`	Name of the workspace. The name is case sensitive.

Usage Notes

The operation fails if any sessions are in `workspace`.

You can unfreeze a workspace only if one or more of the following apply:

- You are the owner of the specified workspace.
- You have the `WM_ADMIN_ROLE`, the `FREEZE_ANY_WORKSPACE` privilege, or the `FREEZE_WORKSPACE` privilege for the specified workspace.

Examples

The following example unfreezes the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.UnfreezeWorkspace ('NEWWORKSPACE');
```

UnlockRows Procedure

Enables access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.UnlockRows(
   workspace        IN VARCHAR2,
   table_name       IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, all_or_user   IN VARCHAR2 DEFAULT 'USER']
   [, lock_mode     IN VARCHAR2 DEFAULT 'ES']);
```

Parameters

Table 80–52 UnlockRows Procedure Parameters

Parameter	Description
workspace	Name of the workspace: locked rows in this workspace and corresponding rows in the parent workspace will be unlocked, as specified in the remaining parameters. The name is case sensitive.
table_name	Name of the table in which rows are to be unlocked. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be unlocked. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are made accessible.
all_or_user	Scope of the request: ALL or USER. ALL: All locks accessible by the user in the specified workspace are considered. USER (default): Only locks owned by the user in the specified workspace are considered.
lock_mode	Locking mode: E, S, or ES. E: Only exclusive mode locks are considered. S: Only shared mode locks are considered. ES (default): Both exclusive mode and shared mode locks are considered.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. For an explanation of Workspace Manager locking, see *Oracle9i Application Developer's Guide - Workspace Manager*.

This procedure unlocks rows that had been previously locked (see the LockRows Procedure). It does not affect whether Workspace Manager locking is set on or off (determined by the SetLockingON Procedure and SetLockingOFF Procedure).

Examples

The following example unlocks the EMPLOYEES table where `last_name = 'Smith'` in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.UnlockRows ('employees', 'NEWWORKSPACE', 'last_name =
''Smith''');
```

81

DBMS_XDB

DBMS_XDB Package contains Resource Management and Access Control APIs for PL/SQL.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB* for more information

This chapter details the following:

- Functions and Procedures of DBMS_XDB

Description of DBMS_XDB

The DBMS_XDB package provides the PL/SQL application developer with APIs that allow resource management in the Oracle XML DB Hierarchy, support for Oracle XML DB's Access Control List (ACL) Security and Oracle XML DB Configuration sessional management.

The Oracle XML DB Resource Management functionality provides `Link()`, `LockResource()`, `GetLockToken()`, `UnlockResource()`, `CreateResource()`, `CreateFolder()`, `DeleteResource()`, `Link()` and functions. These methods complement the functionality provided by Resource Views.

The ACL-based security mechanism can be used with either in-hierarchy ACLs (ACLs stored via the Oracle XML DB resource API) or in-memory ACLs (that may be stored by the user outside Oracle XML DB). Some of these methods can be used for both Oracle XML DB resources and arbitrary database objects.

The Access Control Security functionality provides `checkPrivileges()`, `getAclDocument()`, `changePrivileges()` and `getPrivileges()` functions for Oracle XML DB Resources. `AclCheckPrivileges()` function enables database users access to Oracle XML DB's ACL-based Security mechanism without having to have their objects stored in the Oracle XML DB Hierarchy.

Oracle XML DB Configuration session management provides `CFG_Refresh()`, `CFG_Get()` and `CFG_Update()`.

Functions and Procedures of DBMS_XDB

Table 81-1: Summary of Functions and Procedures of DBMS_XDB

Function/Procedure	Description
getAclDocument() on page 81-3	Retrieves ACL document that protects resource given its pathname.
getPrivileges() on page 81-4	Gets all privileges granted to the current user on the given XDB resource.
changePrivileges() on page 81-4	Adds the given ACE to the given resource's ACL.
checkPrivileges() on page 81-5	Checks access privileges granted to the current user on the specified XDB resource.
setacl() on page 81-6	Sets the ACL on the given XDB resource to be the ACL specified.

Table 81-1: Summary of Functions and Procedures of DBMS_XDB (Cont.)

Function/Procedure	Description
AclCheckPrivileges() on page 81-6	Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter.
LockResource() on page 81-7	Gets a WebDAV-style lock on that resource given a path to that resource.
GetLockToken() on page 81-7	Returns that resource's lock token for the current user given a path to a resource.
UnlockResource() on page 81-8	Unlocks the resource given a lock token and a path to the resource.
CreateResource() on page 81-8	Creates a new resource.
CreateFolder() on page 81-9	Creates a new folder resource in the hierarchy.
DeleteResource() on page 81-10	Deletes a resource from the hierarchy.
Link() on page 81-10	Creates a link to an existing resource.
CFG_Refresh() on page 81-10	Refreshes the session's configuration information to the latest configuration.
CFG_Get() on page 81-11	Retrieves the session's configuration information.
CFG_Update() on page 81-11	Updates the configuration information.

getAclDocument()

Retrieves ACL document that protects resource given its pathname; returns the `xmltype` for ACL document.

Syntax

```
FUNCTION getAclDocument( abspath  IN  VARCHAR2)
                         RETURN sys.xmltype;
```

Parameter	IN / OUT	Description
abspath	(IN)	Pathname of the resource whose ACL doc is required.

getPrivileges()

Gets all privileges granted to the current user on the given XDB resource. Returns an XMLType instance of <privilege> element, which contains the list of all leaf privileges granted on this resource to the current user. For example,

```
<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                              http://xmlns.oracle.com/xdb/acl.xsd"
      <read-contents/>
      <read-properties/>
      <resolve/>
      <read-acl/>
 </privilege>
```

Syntax

```
FUNCTION getPrivileges( res_path IN VARCHAR2) RETURN sys.xmltype;
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy of the XDB resource.

changePrivileges()

Adds the given ACE to the given resource's ACL. Returns positive integer if ACL was successfully modified. For example,

```
<ace  xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dav="DAV:"
          xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                              http://xmlns.oracle.com/xdb/acl.xsd
                              DAV:http://xmlns.oracle.com/xdb/dav.xsd"
     <grant>true</grant>
     <principal>SCOTT</principal>
     <privilege>
          <read-contents/>
          <read-properties/>
          <resolve/>
          <dav:waste/>
     </privilege>
</ace>
```

Syntax

```
FUNCTION changePrivileges( res_path  IN  VARCHAR2,
                           ace       IN  xmltype)
                           RETURN pls_integer;
```

Parameter	IN / OUT	Description
res_path	(IN)	Pathname of the XDB resource for which privileges need to be changed.
ace	(IN)	An XMLType instance of the <ace> element which specifies the <principal>, the operation <grant> and the list of privileges. See the above code example.

If no ACE with the same principal and the same operation (grant / deny) already exists in the ACL, the new ACE is added at the end of the ACL.

checkPrivileges()

Checks access privileges granted to the current user on the specified XDB resource. Returns positive integer if all requested privileges granted. For example, check for <read.contents>, <read.properties> and <dav:waste> privileges using the following <privilege> XMLType instance.

```
<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dav="DAV:"
          xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                              http://xmlns.oracle.com/xdb/acl.xsd
                              DAV: http://xmlns.oracle.com/xdb/dav.xsd"
     <read-contents/>
     <read-properties/>
     <resolve/>
     <dav:waste/>
</privilege>
```

Syntax

```
FUNCTION checkPrivileges( res_path   IN  VARCHAR2,
                          privs      IN  xmltype)
                          RETURN pls_integer;
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy for XDB resource.
privs	(IN)	An XMLType instance of the privilege element specifying the requested set of access privileges. See the above code example.

setacl()

Sets the ACL on the given XDB resource to be the ACL specified by path. The user must have <write-acl> privileges on the resource.

Syntax

```
PROCEDURE setacl( res_path   IN  VARCHAR2,
                  acl_path   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy for XDB resource.
acl_path	(IN)	Absolute path in the Hierarchy for XDB ACL.

AclCheckPrivileges()

Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter. Returns positive integer if all requested privileges granted.

Syntax

```
FUNCTION AclCheckPrivileges( acl_path  IN  VARCHAR2,
                             owner     IN  VARCHAR2,
                             privs     IN  xmltype)
                             RETURN pls_integer;
```

Parameter	IN / OUT	Description
acl_path	(IN)	Absolute path in the Hierarchy for ACL document.
owner	(IN)	Resource owner name; the pseudo user "DAV:owner" is replaced by this user during ACL privilege resolution.

Parameter	IN / OUT	Description
privs	(IN)	An XMLType instance of the privilege element specifying the requested set of access privileges. See description for `checkPrivileges()checkPrivileges()`..

LockResource()

Given a path to a resource, gets a WebDAV-style lock on that resource. Returns TRUE if operation successful; FALSE, otherwise. The user must have UPDATE privileges on the resource.

Syntax

```
FUNCTION LockResource( path      IN  VARCHAR2,
                       depthzero IN  BOOLEAN,
                       shared    IN  boolean)
                       RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to lock.
depthzero	(IN)	CURRENTLY UNSUPPORTED. At this time, only the given resource is locked by this function. In a future release, passing FALSE will obtain an infinite-depth lock.
shared	(IN)	Passing TRUE will obtain a shared write lock.

GetLockToken()

Given a path to a resource, returns that resource's lock token for the current user. The user must have READPROPERTIES privilege on the resource.

Syntax

```
PROCEDURE GetLockToken( path      IN  VARCHAR2,
                        locktoken OUT VARCHAR2);
```

Parameter	IN / OUT	Description
path	(IN)	Path name to the resource.

Parameter	IN / OUT	Description
locktoken	(OUT)	Logged-in user's lock token for the resource.

UnlockResource()

Unlocks the resource given a lock token and a path to the resource. Returns TRUE if operation successful; FALSE, otherwise. The user must have UPDATE privileges on the resource.

Syntax

```
FUNCTION UnlockResource( path     IN  VARCHAR2,
                         deltoken IN  VARCHAR2)
                         RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name to the resource.
deltoken	(IN)	Lock token to be removed.

CreateResource()

Creates a new resource. Returns TRUE if operation successful; FALSE, otherwise.The options are described in the following table.

Syntax	Description
FUNCTION CreateResource(path IN VARCHAR2, data IN VARCHAR2) RETURN BOOLEAN;	Creates a new resource with the given string as its contents.
FUNCTION CreateResource(path IN VARCHAR2, data IN SYS.XMLTYPE) RETURN BOOLEAN;	Creates a new resource with the given XMLType data as its contents.

Syntax	Description
FUNCTION CreateResource(path IN VARCHAR2, datarow IN REF SYS.XMLTYPE) RETURN BOOLEAN;	Given a REF to an existing XMLType row, creates a resource whose contents point to that row. That row should not already exist inside another resource.
FUNCTION CreateResource(path IN VARCHAR2, data IN CLOB) RETURN BOOLEAN;	Creates a resource with the given CLOB as its contents.
FUNCTION CreateResource(path IN VARCHAR2, data IN BFILE) RETURN BOOLEAN;	Creates a resource with the given BFILE as its contents.

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to create. The path name's parent folder must already exist in the hierarchy. In other words, if '/foo/bar.txt' is passed in, then folder '/foo' must already exist.
data	(IN)	The new resource's contents. The data will be parsed to check if it contains a schema-based XML document, and the contents will be stored as schema-based in the schema's default table. Otherwise, it will be saved as binary data.
datarow	(IN)	REF to an XMLType row to be used as the contents.

CreateFolder()

Creates a new folder resource in the hierarchy. Returns TRUE if operation successful; FALSE, otherwise. The given path name's parent folder must already exist in the hierarchy, i.e. if `'/folder1/folder2'` is passed as the path parameter, then `'/folder1'` must already exist.

Syntax

```
FUNCTION CreateFolder( path   IN  VARCHAR2)
                       RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name for the new folder.

DeleteResource()

Deletes a resource from the hierarchy.

Syntax

```
PROCEDURE DeleteResource( path   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to delete.

Link()

Creates a link to an existing resource. This procedures is analogous to creating a hard link in UNIX.

Syntax

```
PROCEDURE Link( srcpath    IN  VARCHAR2,
                linkfolder IN  VARCHAR2,
                linkname   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
srcpath	(IN)	Path name of the resource to which a link is made
linkfolder	(IN)	Folder in which the new link is placed.
linkname	(IN)	Name of the new link.

CFG_Refresh()

Refreshes the session's configuration information to the latest configuration.

Syntax

```
PROCEDURE CFG_Refresh;
```

CFG_Get()

Retrieves the session's configuration information as an XMLType instance.

Syntax

```
FUNCTION CFG_Get RETURN SYS.XMLType;
```

CFG_Update()

Updates the configuration information and commits the change.

Syntax

```
PROCEDURE CFG_Update( xdbconfig   IN  SYS.XMLTYPE);
```

Parameter	IN / OUT	Description
xdbconfig	(IN)	The new configuration data.

82

DBMS_XDBT

The DBMS_XDBT package enables an administrator to create a ConText index on the XML DB hierarchy and configure it for automatic maintenance.

> **See Also:** *Oracle9i XML API Reference - XDK and XDB* for more information

This chapterdetails the following:

- Functions and Procedures of BMS_XDBT

Description of BMS_XDBT

The DBMS_XDBT package provides a convenient mechanism for administrators to set up a ConText index on the Oracle XML DB Hierarchy. The package contains procedures to create default preferences, create the index and set up automatic synchronization of the context index

The DBMS_XDBT package also contains a set of package variables that describe the configuration settings for the index. These are intended to cover the basic customizations that installations may require, but is by no means a complete set.

The DBMS_XDBT package can be used in the following fashion:

- Customize the package to set up the appropriate configuration.
- Drop any existing index preferences using the dropPreferences() procedure.
- Create new index preferences using the createPreferences() procedure.
- Create the ConText index using the createIndex() procedure.
- Set up automatic synchronization of the index using the configureAutoSync() procedure.

Functions and Procedures of BMS_XDBT

Table 82-1: Summary of Functions and Procedures of DBMS_XDBT

Procedure/Function	Description
dropPreferences() on page 82-3	Drops any existing preferences.
createPreferences() on page 82-3	Creates preferences required for the ConText index on the XML DB hierarchy.
createDatastorePref() on page 82-3	Creates a USER datastore preference for the ConText index.
createFilterPref() on page 82-4	Creates a filter preference for the ConText index.
createLexerPref() on page 82-4	Creates a lexer preference for the ConText index.
createWordlistPref() on page 82-4	Creates a stoplist for the ConText index.
createStoplistPref() on page 82-5	Creates a section group for the ConText index.
createStoragePref() on page 82-5	Creates a wordlist preference for the ConText index.
createSectiongroupPref() on page 82-6	Creates a storage preference for the ConText index.

Table 82-1: Summary of Functions and Procedures of DBMS_XDBT (Cont.)

Procedure/Function	Description
createIndex() on page 82-6	Creates the ConText index on the XML DB hierarchy.
configureAutoSync() on page 82-6	Configures the ConText index for automatic maintenance (SYNC).

dropPreferences()

This procedure drops any previously created preferences for the `ConText` index on the XML DB hierarchy.

Syntax

```
PROCEDURE dropPreferences;
```

createPreferences()

This procedure creates a set of default preferences based on the configuration settings.

Syntax

```
PROCEDURE createPreferences;
```

createDatastorePref()

This procedure creates a `USER` datastore preference for the ConText index on the XML DB hierarchy.

- The name of the datastore preference can be modified; see the `DatastorePref` configuration setting.
- The default `USER` datastore procedure also filters the incoming document. The `DBMS_XDBT` package provides a set of configuration settings that control the filtering process.
- The `SkipFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not indexed. Some of the properties of the document metadata, such as author, remain unindexed.
  - The `NullFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not

filtered; however, they are still indexed. This is intended to be used for documents that are text-based, such as HTML, XML and plain-text.

- All other documents use the INSO filter through the IFILTER API.

Syntax

```
PROCEDURE createDatastorePref;
```

createFilterPref()

This procedure creates a NULL filter preference for the ConText index on the XML DB hierarchy.

- The name of the filter preference can be modified; see FilterPref configuration setting.
- The USER datastore procedure filters the incoming document; see createDatastorePref for more details.

Syntax

```
PROCEDURE createFilterPref;
```

createLexerPref()

This procedure creates a BASIC lexer preference for the ConText index on the XML DB hierarchy.

- The name of the lexer preference can be modified; see LexerPref configuration setting. No other configuration settings are provided.
- MultiLexer preferences are not supported.
- Base letter translation is turned on by default.

Syntax

```
PROCEDURE createLexerPref;
```

createWordlistPref()

This procedure creates a wordlist preference for the ConText index on the XML DB hierarchy.

- The name of the wordlist preference can be modified; see the WordlistPref configuration setting. No other configuration settings are provided.
- FUZZY_MATCH and STEMMER attributes are set to AUTO (auto-language detection)

Syntax

```
PROCEDURE createWordlistPref;
```

createStoplistPref()

This procedure creates a stoplist for the ConText index on the XML DB hierarchy.

- The name of the stoplist can be modified; see the StoplistPref configuration setting.
- Numbers are not indexed.
- The StopWords array is a configurable list of stopwords. These are meant to be stopwords in addition to the set of stopwords in CTXSYS.DEFAULT_STOPLIST.

Syntax

```
PROCEDURE createStoplistPref;
```

createStoragePref()

This procedure creates a BASIC_STORAGE preference for the ConText index on the XML DB hierarchy.

- The name of the storage preference can be modified; see the StoragePref configuration setting.
- A tablespace can be specified for the tables and indexes comprising the ConText index; see the IndexTablespace configuration setting.
- Prefix and Substring indexing are not turned on by default.
- The I_INDEX_CLAUSE uses key compression.

Syntax

```
PROCEDURE createStoragePref;
```

createSectiongroupPref()

This procedure creates a section group for the ConText index on the XML DB hierarchy.

- The name of the section group can be changed; see the SectiongroupPref configuration setting.
- The HTML sectioner is used by default. No zone sections are created by default. If the vast majority of documents are XML, consider using the AUTO_SECTION_GROUP or the PATH_SECTION_GROUP; see the SectionGroup configuration setting.

Syntax

```
PROCEDURE createSectiongroupPref;
```

createIndex()

This procedure creates the ConText index on the XML DB hierarchy.

- The name of the index can be changed; see the IndexName configuration setting.
- Set the LogFile configuration parameter to enable ROWID logging during index creation.
- Set the IndexMemory configuration parameter to determine the amount of memory that index creation, and later SYNCs, will use.

Syntax

```
PROCEDURE createIndex;
```

configureAutoSync()

This procedure sets up jobs for automatic SYNCs of the ConText index.

- The system must be configured for job queues for automatic synchronization. The jobs can be viewed using the USER_JOBS catalog views
- The configuration parameter AutoSyncPolicy can be set to choose an appropriate synchronization policy.

The synchronization can be based on one of the following:

Sync Basis	Description
SYNC_BY_PENDING_COUNT	The SYNC is triggered when the number of documents in the pending queue is greater than a threshold (See the MaxPendingCount configuration setting) The pending queue is polled at regular intervals (See the CheckPendingCountInterval configuration parameter) to determine if the number of documents exceeds the threshold
SYNC_BY_TIME	The SYNC is triggered at regular intervals. (See the SyncInterval configuration parameter)
SYNC_BY_PENDING_COUNT_AND_TIME	A combination of both of the above

Syntax

```
PROCEDURE configureAutoSync;
```

Customizing the DBMS_XDBT package

The DBMS_XDBT package can be customized in one of two ways.

- Using a PL/SQL procedure, or an anonymous block, to set the relevant package variables (configuration settings), and then executing the procedures in this package.
- The more general approach is to introduce the appropriate customizations by modifying this package in place, or as a copy.

Please note that the system must be configured to use job queues, and that the jobs can be viewed through the USER_JOBS catalog views.

This section describes the configuration settings, or package variables, available to customize the DBMS_XDBT package.

General Indexing Settings

The following table lists configuration settings that are relevant to general indexing.

Parameter	Default Values	Description
IndexName	XDB$CI	The name of the `ConText` index.
IndexTablespace	XDB$RESINFO	The tablespace used by tables and indexes comprising the `ConText` index.

Parameter	Default Values	Description
IndexMemory	128M	Memory used by index creation and Sync. This must be less than (or equal to) the MAX_INDEX_MEMORY system parameter. The MAX_INDEX_MEMORY system parameter (see the CTX_ADMIN package) must be greater than or equal to the IndexMemory setting.
LogFile	'XdbCtxLog'	The logfile used for ROWID logging during index creation. The LOG_DIRECTORY system parameter must be set already. Set this to NULL to turn off ROWID logging. The LOG_DIRECTORY system parameter (see the CTX_ADMIN package) must be set to enable ROWID logging.

Filtering Settings

The following table lists configuration settings that control the filtering of documents in the XML DB hierarchy.

Parameter	Default Value(s)	Description
SkipFilter_Types	image/%, audio/%, video/%, model/%	List of mime types that should not be indexed.
NullFilter_Types	text/plain, text/html, text/xml	List of mime types that do not need to use the INSO filter. Use this for text-based documents.
FilterPref	XDB$CI_FILTER	Name of the filter preference.

Sectioning and Section Group Settings

The following table lists configuration settings relevant to the sectioner.

Parameter	Default Value	Description
SectionGroup	HTML_SECTION_GROUP	Default sectioner to use. Consider using PATH_SECTION_GROUP or AUTO_SECTION_GROUP if the repository contains mainly XML documents.
SectiongroupPref	XDB$CI_SECTIONGROUP	Name of the section group.

Stoplist Settings

The following table lists stoplist configuration settings.

Parameter	Default Value(s)	Description
StoplistPref	XDB$CI_STOPLIST	Name of the stoplist.
StopWords	0..9 'a'..'z' 'A'..'Z'	List of stopwords, in addition to stopwords specified in CTXSYS.DEFAULT_STOPLIST.

Other Preference Settings

The following table lists settings for other index preferences.

Parameter	Default Value	Description
DatastorePref	XDB$CI_DATASTORE	The name of the datastore preference.
StoragePref	XDB$CI_STORAGE	The name of the storage preference.
WordlistPref	XDB$CI_WORDLIST	The name of the wordlist preference.
DefaultLexerPref	XDB$CI_DEFAULT_ LEXER	The name of the default lexer preference.

Index SYNC settings

The following table lists settings that control when and how the ConText index is synchronized.

Parameter	Default Value(s)	Description
AutoSyncPolicy	SYNC_BY_PENDING_COUNT	Indicates when the index should be SYNCed. Can be one of: - SYNC_BY_PENDING_COUNT, - SYNC_BY_TIME, or - SYNC_BY_PENDING_COUNT_AND_TIME.

Parameter	Default Value(s)	Description
MaxPendingCount	2	Maximum number of documents in the CTX_USER_PENDING queue for this index before an index SYNC is triggered. Applies only if the AutoSyncPolicy is one of: - SYNC_BY_PENDING_COUNT, or - SYNC_BY_PENDING_COUNT_AND_TIME.
CheckPendingCountInterval	10 minutes	Indicates how often, in minutes, the pending queue should be checked. Applies only if the AutoSyncPolicy is one of: - SYNC_BY_PENDING_COUNT, or - SYNC_BY_PENDING_COUNT_AND_TIME.
SyncInterval	60 minutes	Indicates how often, in minutes, the index should be SYNCed. Applies only if the AutoSyncPolicy is one of: - SYNC_BY_TIME, or - SYNC_BY_PENDING_COUNT_AND_TIME

83

DBMS_XDB_VERSION

Oracle XML DB Versioning APIs are found in the DBMS_XBD_VERSION Package.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB* for more information

This chapter details the following:

- Functions and Procedures of DBMS_XDB_VERSION

Description of DBMS_XDB_VERSION

Functions and procedures of DBMS_XDB_VERSION help to create a VCR and manage the versions in the version history.

Functions and Procedures of DBMS_XDB_VERSION

Table 83-1: Summary of Functions and Procedures of DBMS_XDB_VERSION

Function/Procedure	Description
MakeVersioned() on page 83-2	Turns a regular resource whose path name is given into a version-controlled resource.
Checkout() on page 83-3	Checks out a VCR before updating or deleting it.
Checkin() on page 83-3	Checks in a checked-out VCR and returns the resource id of the newly-created version.
Uncheckout() on page 83-4	Checks in a checked-out resource and returns the resource id of the version before the resource is checked out.
GetPredecessors() on page 83-4	Retrieves the list of predecessors by path name.
GetPredsByResId() on page 83-5	Retrieves the list of predecessors by resource id.
GetResourceByResId() on page 83-5	Obtains the resource as an XMLType, given the resource objectID.
GetSuccessors() on page 83-5	Retrieves the list of successors by path name.
GetSuccsByResId() on page 83-6	Retrieves the list of successors by resource id.

MakeVersioned()

Turns a regular resource whose path name is given into a version-controlled resource. If two or more path names are bound with the same resource, a copy of the resource will be created, and the given path name will be bound with the newly-created copy. This new resource is then put under version control. All other path names continue to refer to the original resource. This function returns the resource ID of the first version, or root, of the VCR. This is not an auto-commit SQL operation.

- It is legal to call `MakeVersioned()` for VCR, and neither exception nor warning is raised.

- It is illegal to call `MakeVersioned()` for folder, version history, version resource, and ACL.
- No support for Schema-based resources is provided.

An exception is raised if the resource doesn't exist.

Syntax

```
FUNCTION MakeVersioned( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the resource to be put under version control.

Checkout()

Checks out a VCR before updating or deleting it. This is not an auto-commit SQL operation. Two users of the same workspace cannot `Checkout()` the same VCR at the same time. If this happens, one user must rollback. As a result, it is good practice to commit the `Checkout()` operation before updating a resource and avoid loss of the update if the transaction is rolled back. An exception is raised if the given resource is not a VCR, if the VCR is already checked out, if the resource doesn't exist.

Syntax

```
PROCEDURE Checkout(  pathname VARCHAR2);
```

Parameter	Description
pathname	The path name of the VCR to be checked out.

Checkin()

Checks in a checked-out VCR and returns the resource id of the newly-created version. This is not an auto-commit SQL operation. `Checkin()` doesn't have to take the same path name that was passed to `Checkout()` operation. However, the `Checkin()` path name and the `Checkout()` path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to `Checkin()` because the old name is either invalid or is currently bound with a different resource. Exception is raised if the path name

does not exist. If the path name has been changed, the new path name must be used to `Checkin()` the resource.

Syntax

```
FUNCTION Checkin( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the checked-out resource.

Uncheckout()

Checks in a checked-out resource and returns the resource id of the version before the resource is checked out. This is not an auto-commit SQL operation. `Uncheckout()` doesn't have to take the same path name that was passed to `Checkout()` operation. However, the `Uncheckout()` path name and the `Checkout()` path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to uncheckout because the old name is either invalid or is currently bound with a different resource. An exception is raised if the path name doesn't exist. If the path name has been changed, the new path name must be used to `Checkin()` the resource.

Syntax

```
FUNCTION Uncheckout( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the checked-out resource.

GetPredecessors()

Retrieves the list of predecessors by the path name. An exception is raised if `pathname` is illegal.

Syntax

```
FUNCTION GetPredecessors( pathname VARCHAR2) RETURN resid_list_type;
```

Parameter	Description
pathname	The path name of the resource.

GetPredsByResId()

Retrieves the list of predecessors by resource id. Getting predecessors by `resid` is more efficient than by `pathname`. An exception is raised if the `resid` is illegal.

Syntax

```
FUNCTION GetPredsByResId( resid resid_type) RETURN resid_list_type;
```

Parameter	Description
resid	The resource id.

GetResourceByResId()

Obtains the resource as an XMLType, given the resource objectID. Because the system will not create a path name for versions, this function is useful for retrieving the resource using its resource id.

Syntax

```
FUNCTION GetResourceByResId( resid resid_type) RETURN XMLType;
```

Parameter	Description
resid	The resource id.

GetSuccessors()

Given a version resource or a VCR, retrieves the list of the successors of the resource by the path name. Getting successors by `resid` is more efficient than by `pathname`. An exception is raised if `pathname` is illegal.

Syntax

```
FUNCTION GetSuccessors( pathname VARCHAR2) RETURN resid_list_type;
```

Parameter	Description
pathname	The path name of the resource

GetSuccsByResId()

Given a version resource or a VCR, retrieves the list of the successors of the resource by resource id. Getting successors by resid is more efficient than by path name. An exception is raised if the resid is illegal.

Syntax

```
FUNCTION GetSuccsByResId( resid resid_type) RETURN resid_list_type;
```

Parameter	Description
resid	The resource id.

84

DBMS_XMLDOM

Use DBMS_XMLDOM to access XMLType objects. You can access both schema-based and nonschema-based documents. Before database startup, you must specify the read-from and write-to directories in the initialization.ORA file; for example:

```
UTL_FILE_DIR=/mypath/insidemypath
```

The read-from and write-to files must be on the server file system.

See Also:

- Chapter 95, "UTL_FILE"
- *Oracle9i XML Developer's Kits (XDK) Guide*
- *Oracle9i XML API Reference - XDK and Oracle XML DB*

This chapter details the following:

Description of DBMS_XMLDOM

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense. XML is increasingly being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions. In particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

One important objective of the W3C specification for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language. Since the DOM standard is object-oriented, for this PL/SQL adaptation, some changes had to be made:

- Various DOM interfaces such as Node, Element, etc. have equivalent PL/SQL types DOMNode, DOMElement, etc. respectively.
- Various DOMException codes such as WRONG_DOCUMENT_ERR, HIERARCHY_REQUEST_ERR, etc. have similarly named PL/SQL exceptions
- Various DOM Node type codes such as ELEMENT_NODE, ATTRIBUTE_NODE, etc. have similarly named PL/SQL constants
- Methods defined on a DOM type become functions or procedures that accept it as a parameter. For example, to perform appendChild on a DOM Node n, the `appendChild()` PL/SQL function on page 84-24 is provided,

```
FUNCTION appendChild( n DOMNode,
                      newChild IN DOMNode)
                      RETURN DOMNode;
```

  and to perform setAttribute on a DOM Element elem, the `setAttribute()` PL/SQL procedure on page 84-48 is provided:

```
PROCEDURE setAttribute( elem DOMElement,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

DOM defines an inheritance hierarchy. For example, Document, Element, and Attr are defined to be subtypes of Node. Thus, a method defined in the Node interface should be available in these as well. Since, such inheritance is not directly possible in PL/SQL, the makeNode functions need to be invoked on different DOM types to convert these into a DOMNode. The appropriate functions or procedures that accept DOMNodes can then be called to operate on these types. If, subsequently, type specific functionality is desired, the DOMNode can be converted back into the type by using the `make*()` functions, where DOM* is the desired DOM type.

The implementation of this PL/SQL DOM interface followed the DOM standard of revision REC-DOM-Level-1-19981001. The types and methods described in this document are made available by the PL/SQL package `DBMS_XMLDOM`.

- Before database startup, the read-from and write-to directories in the `initialization.ORA` file must be specified; for example:

```
UTL_FILE_DIR=/mypath/insidemypath
```

- The read-from and write-to files must be on the server file system.

Types of DBMS_XMLDOM

The following types for DBMS_XMLDOM.DOMTYPE are defined in Table 84-1:

Table 84-1: XDB_XMLDOM Types

Type	Description
DOMNode	Implements the DOM Node interface.
DOMNamedNodeMap	Implements the DOM NamedNodeMap interface.
DOMNodeList	Implements the DOM NodeList interface.
DOMAttr	Implements the DOM Attribute interface.
DOMCDataSection	Implements the DOM CDataSection interface.
DOMCharacterData	Implements the DOM Character Data interface.
DOMComment	Implements the DOM Comment interface.
DOMDocumentFragment	Implements the DOM DocumentFragment interface.
DOMElement	Implements the DOM Element interface.
DOMEntity	Implements the DOM Entity interface.

Table 84-1: XDB_XMLDOM Types (Cont.)

Type	Description
DOMEntityReference	Implements the DOM EntityReference interface.
DOMNotation	Implements the DOM Notation interface.
DOMProcessingInstruction	Implements the DOM Processing instruction interface.
DOMText	Implements the DOM Text interface.
DOMImplementation	Implements the DOM DOMImplementation interface.
DOMDocumentType	Implements the DOM Document Type interface.
DOMDocument	Implements the DOM Document interface.

Defined Constants of DBMS_XMLDOM

The constants listed in Table 84-2 are defined for `DBMS_XMLDOM`. For example, when a request such as `getNodeType(myNode)` is made, the returned type will be one of the following constants.

Table 84-2: Defined Constants for DBMS_XMLDDOM

Constant	Description
ELEMENT_NODE	The Node is an Element.
ATTRIBUTE_NODE	The Node is an Attribute.
TEXT_NODE	The Node is a Text node.
CDATA_SECTION_NODE	The Node is a CDataSection.
ENTITY_REFERENCE_NODE	The Node is an Entity Reference.
ENTITY_NODE	The Node is an Entity.
PROCESSING_INSTRUCTION_NODE	The Node is a Processing Instruction.
COMMENT_NODE	The Node is a Comment.
DOCUMENT_NODE	The Node is a Document.
DOCUMENT_TYPE_NODE	The Node is a Document Type Definition.
DOCUMENT_FRAGMENT_NODE	The Node is a Document fragment.
NOTATION_NODE	The Node is a Notation.

Exceptions of DBMS_XMLDOM

The exceptions listed in Table 84-3 are defined for DBMS_XMLDOM:

Table 84-3: Exceptions for DBMS_XMLDDOM

Exception	Description
INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value.
DOMSTRING_SIZE_ERR	If the specified range of text does not fit into a DOMString.
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong.
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it).
INVALID_CHARACTER_ERR	If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.
NO_DATA_ALLOWED_ERROR	If data is specified for a node which does not support data.
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed.
NO_FOUND_ERR	If an attempt is made to reference a node in a context where it does not exist.
NOT_SUPPORTED_ERR	If the implementation does not support the requested type of object or operation.
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already in use elsewhere.

Functions and Procedures of DBMS_XMLDOM

DBMS_XMLDOM subprograms are divided into groups according to w3c Interfaces.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
DOM Node Methods	
isNull() on page 84-12	Tests if the node is NULL.
makeAttr() on page 84-12	Casts the node to an Attribute.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
makeCDataSection() on page 84-13	Casts the node to a CDataSection.
makeCharacterData() on page 84-13	Casts the node to CharacterData.
makeComment() on page 84-13	Casts the node to a Comment.
makeDocumentFragment() on page 84-14	Casts the node to a DocumentFragment.
makeDocumentType() on page 84-14	Casts the node to a Document Type.
makeElement() on page 84-14	Casts the node to an Element.
makeEntity() on page 84-14	Casts the node to an Entity.
makeEntityReference() on page 84-15	Casts the node to an EntityReference.
makeNotation() on page 84-15	Casts the node to a Notation.
makeProcessingInstruction() on page 84-15	Casts the node to a DOMProcessingInstruction.
makeText() on page 84-16	Casts the node to a DOMText.
makeDocument() on page 84-16	Casts the node to a DOMDocument.
writeToFile() on page 84-16	Writes the contents of the node to a file.
writeToBuffer() on page 84-17	Writes the contents of the node to a buffer.
writeToClob() on page 84-17	Writes the contents of the node to a clob.
getNodeName() on page 84-18	Retrieves the Name of the Node.
getNodeValue() on page 84-18	Retrieves the Value of the Node.
setNodeValue() on page 84-19	Sets the Value of the Node.
getNodeType() on page 84-19	Retrieves the Type of the node.
getParentNode() on page 84-19	Retrieves the parent of the node.
getChildNodes() on page 84-20	Retrieves the children of the node.
getFirstChild() on page 84-20	Retrieves the first child of the node.
getLastChild() on page 84-20	Retrieves the last child of the node.
getPreviousSibling() on page 84-21	Retrieves the previous sibling of the node.
getNextSibling() on page 84-21	Retrieves the next sibling of the node.
getAttributes() on page 84-21	Retrieves the attributes of the node.
getOwnerDocument() on page 84-22	Retrieves the owner document of the node.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
insertBefore() on page 84-22	Inserts a child before the reference child.
replaceChild() on page 84-23	Replaces the old child with a new child.
removeChild() on page 84-23	Removes a specified child from a node.
appendChild() on page 84-24	Appends a new child to the node.
hasChildNodes() on page 84-24	Tests if the node has child nodes.
cloneNode() on page 84-24	Clones the node.
DOM Named Node Map methods	
isNull() on page 84-25	Tests if the NamedNodeMap is NULL.
getNamedItem() on page 84-25	Retrieves the item specified by the name.
setNamedItem() on page 84-26	Sets the item in the map specified by the name.
removeNamedItem() on page 84-26	Removes the item specified by name.
item() on page 84-27	Retrieves the item given the index in the map.
getLength() on page 84-27	Retrieves the number of items in the map.
DOM Node List Methods	
isNull() on page 84-28	Tests if the Nodelist is NULL.
item() on page 84-28	Retrieves the item given the index in the nodelist.
getLength() on page 84-28	Retrieves the number of items in the list.
DOM Attr Methods	
isNull() on page 84-29	Tests if the Attribute Node is NULL.
makeNode() on page 84-29	Casts the Attribute to a node.
getQualifiedName() on page 84-29	Retrieves the Qualified Name of the attribute.
getNamespace() on page 84-30	Retrieves the NS URI of the attribute.
getLocalName() on page 84-30	Retrieves the local name of the attribute.
getExpandedName() on page 84-30	Retrieves the expanded name of the attribute.
getName() on page 84-31	Retrieves the name of the attribute.
getSpecified() on page 84-31	Tests if attribute was specified in the owning element.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
getValue() on page 84-31	Retrieves the value of the attribute.
setValue() on page 84-31	Sets the value of the attribute.
DOM C Data Section Methods	
isNull() on page 84-32	Tests if the CDataSection is NULL.
makeNode() on page 84-32	Casts the CDatasection to a node.
DOM Character Data Methods	
isNull() on page 84-33	Tests if the CharacterData is NULL.
makeNode() on page 84-33	Casts the CharacterData to a node.
getData() on page 84-33	Retrieves the data of the node.
setData() on page 84-34	Sets the data to the node.
getLength() on page 84-34	Retrieves the length of the data.
substringData() on page 84-34	Retrieves the substring of the data.
appendData() on page 84-35	Appends the given data to the node data.
insertData() on page 84-35	Inserts the data in the node at the given offSets.
deleteData() on page 84-36	Deletes the data from the given offSets.
replaceData() on page 84-36	Replaces the data from the given offSets.
DOM Comment Methods	
isNull() on page 84-37	Tests if the comment is NULL.
makeNode() on page 84-37	Casts the Comment to a node.
DOM Implementation Methods	
isNull() on page 84-37	Tests if the DOMImplementation node is NULL.
hasFeature() on page 84-38	Tests if the DOMImplementation implements a given feature.
DOM Document Fragment Methods	
isNull() on page 84-38	Tests if the DocumentFragment is NULL.
makeNode() on page 84-38	Casts the Document Fragment to a node.
DOM Document Type Methods	

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
isNull() on page 84-39	Tests if the Document Type is NULL.
makeNode() on page 84-39	Casts the document type to a node.
findEntity() on page 84-39	Finds the specified entity in the document type.
findNotation() on page 84-40	Finds the specified notation in the document type.
getPublicId() on page 84-40	Retrieves the public ID of the document type.
getSystemId() on page 84-41	Retrieves the system ID of the document type.
writeExternalDTDToFile() on page 84-41	Writes the document type definition to a file.
writeExternalDTDToBuffer() on page 84-41	Writes the document type definition to a buffer.
writeExternalDTDToClob() on page 84-42	Writes the document type definition to a clob.
getName() on page 84-43	Retrieves the name of the Document type.
getEntities() on page 84-43	Retrieves the nodemap of entities in the Document type.
getNotations() on page 84-43	Retrieves the nodemap of the notations in the Document type.
DOM Element Methods	
isNull() on page 84-44	Tests if the Element is NULL.
makeNode() on page 84-44	Casts the Element to a node.
getQualifiedName() on page 84-44	Retrieves the qualified name of the element.
getNamespace() on page 84-45	Retrieves the NS URI of the element.
getLocalName() on page 84-45	Retrieves the local name of the element.
getExpandedName() on page 84-45	Retrieves the expanded name of the element.
getChildrenByTagName() on page 84-45	Retrieves the children of the element by tag name.
getElementsByTagName() on page 84-46	Retrieves the elements in the subtree by tagname.
resolveNamespacePrefix() on page 84-47	Resolve the prefix to a namespace uri.
getTagName() on page 84-47	Retrieves the Tag name of the element.
getAttribute() on page 84-47	Retrieves the attribute node specified by the name.
setAttribute() on page 84-48	Sets the attribute specified by the name.
removeAttribute() on page 84-48	Removes the attribute specified by the name.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
getAttributeNode() on page 84-49	Retrieves the attribute node specified by the name.
setAttributeNode() on page 84-49	Sets the attribute node in the element.
removeAttributeNode() on page 84-49	Removes the attribute node in the element.
normalize() on page 84-50	Normalizes the text children of the element.
DOM Entity Methods	
isNull() on page 84-50	Tests if the Entity is NULL.
makeNode() on page 84-50	Casts the Entity to a node.
getPublicId() on page 84-51	Retrieves the public Id of the entity.
getSystemId() on page 84-51	Retrieves the system Id of the entity.
getNotationName() on page 84-51	Retrieves the notation name of the entity.
DOM Entity Reference Methods	
isNull() on page 84-52	Tests if the Entity Reference is NULL.
makeNode() on page 84-52	Casts the Entity Reference to NULL.
DOM Notation Methods	
isNull() on page 84-52	Tests if the Notation is NULL.
makeNode() on page 84-53	Casts the notation to a node.
getPublicId() on page 84-53	Retrieves the public Id of the notation.
getSystemId() on page 84-53	Retrieves the system Id of the notation.
DOM Processing Instruction Methods	
isNull() on page 84-54	Tests if the Processing Instruction is NULL.
makeNode() on page 84-54	Casts the Processing Instruction to a node.
getData() on page 84-54	Retrieves the data of the processing instruction.
getTarget() on page 84-55	Retrieves the target of the processing instruction.
setData() on page 84-55	Sets the data of the processing instruction.
DOM Text Methods	
isNull() on page 84-55	Tests if the text is NULL.
makeNode() on page 84-56	Casts the text to a node.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
splitText() on page 84-56	Splits the contents of the text node into 2 text nodes.
DOM Document Methods	
isNull() on page 84-56	Tests if the document is NULL.
makeNode() on page 84-57	Casts the document to a node.
newDOMDocument() on page 84-57	Creates a new Document.
freeDocument() on page 84-57	Frees the document.
getVersion() on page 84-58	Retrieves the version of the document.
setVersion() on page 84-58	Sets the version of the document.
getCharset() on page 84-58	Retrieves the Character set of the document.
setCharset() on page 84-58	Sets the Character set of the document.
getStandalone() on page 84-59	Retrieves if the document is specified as standalone.
setStandalone() on page 84-59	Sets the document standalone.
writeToFile() on page 84-59	Writes the document to a file.
writeToBuffer() on page 84-60	Writes the document to a buffer.
writeToClob() on page 84-61	Writes the document to a clob.
writeExternalDTDToFile() on page 84-61	Writes the DTD of the document to a file.
writeExternalDTDToBuffer() on page 84-62	Writes the DTD of the document to a buffer.
writeExternalDTDToClob() on page 84-62	Writes the DTD of the document to a clob.
getDoctype() on page 84-63	Retrieves the DTD of the document.
getImplementation() on page 84-63	Retrieves the DOM implementation.
getDocumentElement() on page 84-63	Retrieves the root element of the document.
createElement() on page 84-64	Creates a new Element.
createDocumentFragment() on page 84-64	Creates a new Document Fragment.
createTextNode() on page 84-64	Creates a Text node.
createComment() on page 84-65	Creates a Comment node.
createCDATASection() on page 84-65	Creates a CDatasection node.
createProcessingInstruction() on page 84-65	Creates a Processing Instruction.

Table 84-4: Summary of Functions and Procedures of DBMS_XMLDOM

Group/Method	Description
createAttribute() on page 84-66	Creates an Attribute.
createEntityReference() on page 84-66	Creates an Entity reference.
getElementsByTagName() on page 84-67	Retrieves the elements in the by tag name.

DOM Node Methods

isNull()

Checks if the given DOMNode is NULL. Returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( n DOMNode) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to check.

makeAttr()

Casts a given DOMNode to a DOMAttr, and returns the DOMAttr.

Syntax

```
FUNCTION makeAttr( n DOMNode) RETURN DOMAttr;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeCDataSection()

Casts a given DOMNode to a DOMCDataSection, and returns the DOMCDataSection.

Syntax

```
FUNCTION makeCDataSection( n DOMNode) RETURN DOMCDataSection;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeCharacterData()

Casts a given DOMNode to a DOMCharacterData, and returns the DOMCharacterData.

Syntax

```
FUNCTION makeCharacterData( n DOMNode) RETURN DOMCharacterData;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeComment()

Casts a given DOMNode to a DOMComment, and returns the DOMComment.

Syntax

```
FUNCTION makeComment(n DOMNode) RETURN DOMComment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocumentFragment()

Casts a given DOMNode to a DOMDocumentFragment, and returns the DOMDocumentFragment.

Syntax

```
FUNCTION makeDocumentFragment( n DOMNode) RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocumentType()

Casts a given DOMNode to a DOMDocumentType and returns the DOMDocumentType.

Syntax

```
FUNCTION makeDocumentType(n DOMNode) RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeElement()

Casts a given DOMNode to a DOMElement, and returns the DOMElement.

Syntax

```
FUNCTION makeElement(n DOMNode) RETURN DOMElement;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeEntity()

Casts a given DOMNode to a DOMEntity, and returns the DOMEntity.

Syntax

```
FUNCTION makeEntity(n DOMNode) RETURN DOMEntity;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeEntityReference()

Casts a given DOMNode to a DOMEntityReference, and returns the DOMEntityReference.

Syntax

```
FUNCTION makeEntityReference(n DOMNode) RETURN DOMEntityReference;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeNotation()

Casts a given DOMNode to a DOMNotation, and returns the DOMNotation.

Syntax

```
FUNCTION makeNotation(n DOMNode) RETURN DOMNotation;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeProcessingInstruction()

Casts a given DOMNode to a DOMProcessingInstruction, and returns the DOMProcessingInstruction.

Syntax

```
FUNCTION makeProcessingInstruction( n DOMNode)
                                    RETURN DOMProcessingInstruction;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeText()

Casts a given DOMNode to a DOMText, and returns the DOMText.

Syntax

```
FUNCTION makeText(n DOMNode) RETURN DOMText;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocument()

Casts a given DOMNode to a DOMDocument, and returns the DOMDocument.

Syntax

```
FUNCTION makeDocument(n DOMNode) RETURN DOMDocument;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

writeToFile()

Writes XML node to specified file. The options are given in the table below.

Syntax	Description
PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2);	Writes XML node to specified file using the database character set.

Syntax	Description
PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2, charset VARCHAR2);	Writes XML node to specified file using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
fileName	(IN)	File to write to.
charset	(IN)	Given character set.

writeToBuffer()

Writes XML node to specified buffer. The options are given in the table below.

Syntax	Description
PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2);	Writes XML node to specified buffer using the database character set.
PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes XML node to specified buffer using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
buffer	(IN/OUT)	Buffer to write to.
charset	(IN)	Given character set.

writeToClob()

Writes XML node to specified clob. The options are given in the table below.

Syntax	Description
PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB);	Writes XML node to specified clob using the database character set.
PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB, charset VARCHAR2);	Writes XML node to specified clob using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
cl	(IN/OUT)	CLOB to write to.
charset	(IN)	Given character set.

getNodeName()

Get the name of the node depending on its type

Syntax

```
FUNCTION getNodeName(n DOMNode) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getNodeValue()

Get the value of this node, depending on its type.

Syntax

```
FUNCTION getNodeValue(n DOMNode) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	IN	DOMNode.

setNodeValue()

Sets the value of this node, depending on its type. When it is defined to be null, setting it has no effect.

Syntax

```
PROCEDURE setNodeValue( n DOMNode,
                        nodeValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
n	IN	DOMNode.
nodeValue	IN	The value to which node is set.

getNodeType()

Retrieves a code representing the type of the underlying object.

Syntax

```
FUNCTION getNodeType(n DOMNode) RETURN NUMBER;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getParentNode()

Retrieves the parent of this node. All nodes, except Attr, Document, DocumentFragment, Entity, and Notation may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Syntax

```
FUNCTION getParentNode(n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getChildNodes()

Retrieves a NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

Syntax

```
FUNCTION getChildNodes(n DOMNode) RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getFirstChild()

Retrieves the first child of this node. If there is no such node, this returns null.

Syntax

```
FUNCTION getFirstChild( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getLastChild()

Retrieves the last child of this node. If there is no such node, this returns NULL.

Syntax

```
FUNCTION getLastChild( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getPreviousSibling()

Retrieves the node immediately preceding this node. If there is no such node, this returns NULL.

Syntax

```
FUNCTION getPreviousSibling( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getNextSibling()

Retrieves the node immediately following this node. If there is no such node, this returns NULL.

Syntax

```
FUNCTION getNextSibling( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getAttributes()

Retrieves a NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Syntax

```
FUNCTION getAttributes( n DOMNode) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getOwnerDocument()

Retrieves the Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a DocumentType which is not used with any Document yet, this is null.

Syntax

```
FUNCTION getOwnerDocument( n DOMNode) RETURN DOMDocument;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

insertBefore()

Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is `NULL`, insert `newChild` at the end of the list of children.

If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed.

Syntax

```
FUNCTION insertBefore( n DOMNode,
                       newChild IN DOMNode,
                       refChild IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The child to be inserted in the DOMNode n
refChild	IN	The reference node before which the newChild is to be inserted

replaceChild()

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the oldChild node.

If `newChild` is a DocumentFragment object, `oldChild` is replaced by all of the DocumentFragment children, which are inserted in the same order. If the newChild is already in the tree, it is first removed.

Syntax

```
FUNCTION replaceChild( n DOMNode,
                       newChild IN DOMNode,
                       oldChild IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The new Child which is to replace the oldChild
oldChild	IN	The child of the Node n which is to be replaced

removeChild()

Removes the child node indicated by oldChild from the list of children, and returns it.

Syntax

```
FUNCTION removeChild( n DOMNode,
                      oldChild IN DOMNode)
                      RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
oldCHild	IN	The child of the node n to be removed

appendChild()

Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Syntax

```
FUNCTION appendChild( n DOMNode,
                      newChild IN DOMNode)
                      RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The child to be appended to the list of children of Node n

hasChildNodes()

Returns whether this node has any children.

Syntax

```
FUNCTION hasChildNodes( n DOMNode) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

cloneNode()

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent; its `parentNode` is `NULL`.

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is true). Cloning any other type of node simply returns a copy of this node.

Note that cloning an immutable subtree results in a mutable copy, but the children of an EntityReference clone are read-only. In addition, clones of unspecified Attr

nodes are specified. And, cloning Document, DocumentType, Entity, and Notation nodes is implementation dependent.

Syntax

```
FUNCTION cloneNode( n DOMNode,
                    deep boolean)
                    RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
deep	IN	boolean to determine if children are to be cloned or not

DOM Named Node Map Methods

isNull()

Checks that the given DOMNamedNodeMap is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( nnm DOMNamedNodeMap) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
nnm	(IN)	DOMNameNodeMap to check.

getNamedItem()

Retrieves a node specified by name.

Syntax

```
FUNCTION getNamedItem( nnm DOMNamedNodeMap,
                       name IN VARCHAR2)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
name	IN	Name of the item to be retrieved

setNamedItem()

Adds a node using its nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one.

As the nodeName attribute is used to derive the name under which the node must be stored, multiple nodes of certain types, those that have a "special" string value, cannot be stored because the names would clash. This is seen as preferable to allowing nodes to be aliased.

Syntax

```
FUNCTION setNamedItem( nnm DOMNamedNodeMap,
                       arg IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
arg	IN	The Node to be added using its NodeName attribute.

removeNamedItem()

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

Syntax

```
FUNCTION removeNamedItem( nnm DOMNamedNodeMap,
                          name IN VARCHAR2)
                          RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
name	IN	The name of the item to be removed from the map

item()

Returns the item in the map which corresponds to the `index` parameter. If index is greater than or equal to the number of nodes in this map, this returns `NULL`.

Syntax

```
FUNCTION item( nnm DOMNamedNodeMap,
               index IN NUMBER)
               RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
index	IN	The index in the node map at which the item is to be retrieved

getLength()

The number of nodes in this map. The range of valid child node indices is 0 to length-1 inclusive.

Syntax

```
FUNCTION getLength( nnm DOMNamedNodeMap) RETURN NUMBER;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap

DOM Node List Methods

isNull()

Checks that the given DOMNodeList is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( nl DOMNodeList) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
nl	(IN)	DOMNodeList to check.

item()

Returns the item in the collection which corresponds to the index parameter. If index is greater than or equal to the number of nodes in the list, this returns null.

Syntax

```
FUNCTION item( nl DOMNodeList,
               index IN NUMBER)
               RETURN DOMNode;
```

Parameter	IN / OUT	Description
nl	IN	DOMNodeList
index	IN	The index in the nodelist at which to retrieve the item from

getLength()

The number of nodes in the list. The range of valid child node indices is 0 to length-1 inclusive.

Syntax

```
FUNCTION getLength( nl DOMNodeList) RETURN NUMBER;
```

Parameter	IN / OUT	Description
nl	IN	DOMNodeList

DOM Attr Methods

isNull()

Checks that the given DOMAttr is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( a DOMAttr) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr to check.

makeNode()

Casts given DOMAttr to a DOMNode, and returns the DOMNode.

Syntax

```
FUNCTION makeNode( a DOMAttr) RETURN DOMNode;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr to cast.

getQualifiedName()

Returns the qualified name of the DOMAttr.

Syntax

```
FUNCTION getQualifiedName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getNamespace()

Returns the namespace of the DOMAttr.

Syntax

```
FUNCTION getNamespace( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getLocalName()

Returns the local name of the DOMAttr.

Syntax

```
FUNCTION getLocalName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getExpandedName()

Returns the expanded name of the DOMAttr.

Syntax

```
FUNCTION getExpandedName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getName()

Returns the name of this attribute.

Syntax

```
FUNCTION getName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

getSpecified()

If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false.

Syntax

```
FUNCTION getSpecified( a DOMAttr) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

getValue()

Retrieves the value of the attribute.

Syntax

```
FUNCTION getValue( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

setValue()

Sets the value of the attribute.

Syntax

```
PROCEDURE setValue( a DOMAttr,
                    value IN VARCHAR2);
```

Parameter	IN / OUT	Description
a	IN	DOMAttr
value	IN	The value to set the attribute to

DOM C Data Section Methods

isNull()

Checks that the given DOMCDataSection is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( cds DOMCDataSection) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
cds	(IN)	DOMCDataSection to check.

makeNode()

Casts the DOMCDataSection to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( cds DOMCDataSection) RETURN DOMNode;
```

Parameter	IN / OUT	Description
cds	(IN)	DOMCDataSection to cast.

Character Data Methods

isNull()

Checks that the given DOMCharacterData is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( cd DOMCharacterData) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
cd	(IN)	DOMCharacterData to check.

makeNode()

Casts the given DOMCharacterData as a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( cd DOMCharacterData) RETURN DOMNode;
```

Parameter	IN / OUT	Description
cd	(IN)	DOMCharacterData to cast

getData()

Gets the character data of the node that implements this interface.

Syntax

```
FUNCTION getData( cd DOMCharacterData) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData

setData()

Sets the character data of the node that implements this interface.

Syntax

```
PROCEDURE setData( cd DOMCharacterData,
                   data IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData
data	IN	The data to set the node to

getLength()

The number of 16-bit units that are available through data and the `substringData()` method. This may have the value zero, i.e., CharacterData nodes may be empty.

Syntax

```
FUNCTION getLength( cd DOMCharacterData) RETURN NUMBER;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData

substringData()

Extracts a range of data from the node.

Syntax

```
FUNCTION substringData( cd DOMCharacterData,
                        offset IN NUMBER,
                        cnt IN NUMBER)
                        RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The starting offset of the data from which to get the data.
cnt	IN	The number of characters (from the offset) of the data to get.

appendData()

Appends the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the specified string argument.

Syntax

```
PROCEDURE appendData( cd DOMCharacterData,
                      arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
arg	IN	The data to append to the existing data.

insertData()

Inserts a string at the specified 16-bit unit offset.

Syntax

```
PROCEDURE insertData( cd DOMCharacterData,
                      offset IN NUMBER,
                      arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The offset at which to insert the data.
arg	IN	The value to be inserted.

deleteData()

Removes a range of 16-bit units from the node. Upon success, data and length reflect the change.

Syntax

```
PROCEDURE deleteData( cd DOMCharacterData,
                      offset IN NUMBER,
                      cnt IN NUMBER);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData
offset	IN	The offset from which to delete the data
cnt	IN	The number of characters (starting from offset) to delete.

replaceData()

Remove a range of 16-bit units from the node. Upon success, data and length reflect the change.

Syntax

```
PROCEDURE replaceData( cd DOMCharacterData,
                       offset IN NUMBER,
                       cnt IN NUMBER,
                       arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The offset at which to replace.
cnt	IN	The no. of characters to replace.
arg	IN	The value to replace with.

DOM Comment Methods

isNull()

Checks that the given DOMComment is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( com DOMComment) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
com	(IN)	DOMComment to check.

makeNode()

Casts the given DOMComment to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( com DOMComment) RETURN DOMNode;
```

Parameter	IN / OUT	Description
com	(IN)	DOMComment to cast.

DOM Implementation Methods

isNull()

Checks that the given DOMImplementation is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( di DOMImplementation) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
di	(IN)	DOMImplementation to check.

hasFeature()

Test if the DOM implementation implements a specific feature.

Syntax

```
FUNCTION hasFeature( di DOMImplementation,
                    feature IN VARCHAR2,
                    version IN VARCHAR2)
                    RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
di	IN	DOMImplementation
feature	IN	The feature to check for
version	IN	The version of the DOM to check in

DOM Document Fragment Methods

isNull()

Checks that the given DOMDocumentFragment is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( df DOMDocumentFragment) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
df	(IN)	DOMDocumentFragment to check.

makeNode()

Casts the given DOMDocumentFragment to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( df DOMDocumentFragment) RETURN DOMNode;
```

Parameter	IN / OUT	Description
df	(IN)	DOMDocumentFragment to cast.

DOM Document Type Methods

isNull()

Checks that the given DOMDocumentType is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( dt DOMDocumentType) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
dt	(IN)	DOMDocumentType to check.

makeNode()

Casts the given DOMDocumentType to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( dt DOMDocumentType) RETURN DOMNode;
```

Parameter	IN / OUT	Description
dt	(IN)	DOMDocumentType to cast.

findEntity()

Finds an entity in the given DTD; returns that entity if found.

Syntax

```
FUNCTION findEntity( dt      DOMDocumentType,
                     name    VARCHAR2,
                     par     BOOLEAN)
                     RETURN  DOMEntity;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
name	(IN)	Entity to find.
par	(IN)	Flag to indicate type of entity; TRUE for parameter entity and FALSE for normal entity.

findNotation()

Finds the notation in the given DTD; returns it, if found.

Syntax

```
FUNCTION findNotation( dt   DOMDocumentType,
                      name VARCHAR2)
                      RETURN DOMNotation;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
name	(IN)	The notation to find.

getPublicId()

Returns the public id of the given DTD.

Syntax

```
FUNCTION getPublicId( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.

getSystemId()

Returns the system id of the given DTD.

Syntax

```
FUNCTION getSystemId( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.

writeExternalDTDToFile()

Writes DTD to a specified file. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToFile(dt DOMDocumentType, fileName VARCHAR2);	Writes the DTD to a specified file using the database character set.
PROCEDURE writeExternalDTDToFile(dt DOMDocumentType, fileName VARCHAR2, charset VARCHAR2);	Writes the DTD to a specified file using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
fileName	(IN)	The file to write to.
charset	(IN)	Character set.

writeExternalDTDToBuffer()

Writes DTD to a specified buffer. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToBuffer(dt DOMDocumentType, buffer IN OUT VARCHAR2);	Writes the DTD to a specified buffer using the database character set.
PROCEDURE writeExternalDTDToBuffer(dt DOMDocumentType, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes the DTD to a specified buffer using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
buffer	(IN/OUT)	The buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToClob()

Writes DTD to a specified clob. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToClob(dt DOMDocumentType, cl IN OUT CLOB);	Writes the DTD to a specified clob using the database character set.
PROCEDURE writeExternalDTDToClob(dt DOMDocumentType, cl IN OUT CLOB, charset VARCHAR2);	Writes the DTD to a specified clob using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
cl	(IN/OUT)	The clob to write to.
charset	(IN)	Character set.

getName()

Retrieves the name of DTD, or the name immediately following the DOCTYPE keyword.

Syntax

```
FUNCTION getName( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

getEntities()

Retrieves a NamedNodeMap containing the general entities, both external and internal, declared in the DTD.

Syntax

```
FUNCTION getEntities( dt DOMDocumentType) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

getNotations()

Retrieves a NamedNodeMap containing the notations declared in the DTD.

Syntax

```
FUNCTION getNotations( dt DOMDocumentType) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

DOM Element Methods

isNull()

Checks that the given DOMElement is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( elem DOMElement) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement to check.

makeNode()

Casts the given DOMElement to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( elem DOMElement) RETURN DOMNode;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement to cast.

getQualifiedName()

Returns the qualified name of the DOMElement.

Syntax

```
FUNCTION getQualifiedName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getNamespace()

Returns the namespace of the DOMElement.

Syntax

```
FUNCTION getNamespace( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getLocalName()

Returns the local name of the DOMElement.

Syntax

```
FUNCTION getLocalName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getExpandedName()

Returns the expanded name of the DOMElement.

Syntax

```
FUNCTION getExpandedName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getChildrenByTagName()

Returns the children of the DOMElement. The options are given in the table below.

Syntax	Description
FUNCTION getChildrenByTagName(elem DOMElement, name IN VARCHAR2) RETURN DOMNodeList;	Returns children of the DOMElement given the tag name.
FUNCTION getChildrenByTagName(elem DOMElement, name IN VARCHAR2, ns VARCHAR2) RETURN DOMNodeList;	Returns children of the DOMElement given the tag name and namespace.

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Tag name; * matches any tag.
ns	(IN)	Namespace.

getElementsByTagName()

Returns the element children of the DOMElement. The options are given in the table below.

Syntax	Description
FUNCTION getElementsByTagName(elem DOMElement, name IN VARCHAR2) RETURN DOMNodeList;	Returns the element children of the DOMElement given the tag name.
FUNCTION getElementsByTagName(elem DOMElement, name IN VARCHAR2, ns VARCHAR2) RETURN DOMNodeList;	Returns the element children of the DOMElement given the tag name and namespace.

Parameters

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Tag name; * matches any tag.
ns	(IN)	Namespace.

resolveNamespacePrefix()

Resolves the given namespace prefix, and returns the resolved namespace.

Syntax

```
FUNCTION resolveNamespacePrefix( elem DOMElement,
                                 prefix VARCHAR2)
                                 RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
prefix	(IN)	Namespace prefix.

getTagName()

Returns the name of the DOMElement.

Syntax

```
FUNCTION getTagName(elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

getAttribute()

Returns the value of a DOMElement's attribute by name.

Syntax

```
FUNCTION getAttribute( elem DOMElement,
                       name IN VARCHAR2)
                       RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.

setAttribute()

Sets the value of a DOMElement's attribute by name.

Syntax

```
PROCEDURE setAttribute( elem DOMElement,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.
value	(IN)	Attribute value

removeAttribute()

Removes an attribute from the DOMElement by name.

Syntax

```
PROCEDURE removeAttribute( elem DOMElement,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

Parameter	IN / OUT	Description
name	(IN)	Attribute name; * matches any attribute.

getAttributeNode()

Returns an attribute node from the DOMElement by name.

Syntax

```
FUNCTION getAttributeNode( elem DOMElement,
                           name IN VARCHAR2)
                           RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.

setAttributeNode()

Adds a new attribute node to the DOMElement.

Syntax

```
FUNCTION setAttributeNode( elem DOMElement,
                           newAttr IN DOMAttr)
                           RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
newAttr	(IN)	The new DOMAttr.

removeAttributeNode()

Removes the specified attribute node from the DOMElement.

Syntax

```
FUNCTION removeAttributeNode( elem DOMElement,
```

```
                                        oldAttr IN DOMAttr)
                                        RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
oldAttr	(IN)	The old DOMAttr.

normalize()

Normalizes the text children of the DOMElement.

Syntax

```
PROCEDURE normalize( elem DOMElement);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

DOM Entity Methods

isNull()

Checks that the given DOMEntity is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( ent DOMEntity) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity to check.

makeNode()

Casts given DOMEntity to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( ent DOMEntity) RETURN DOMNode;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity to cast.

getPublicId()

Returns the public identifier of the DOMEntity.

Syntax

```
FUNCTION getPublicId( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

getSystemId()

Returns the system identifier of the DOMEntity.

Syntax

```
FUNCTION getSystemId( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

getNotationName()

Returns the notation name of the DOMEntity.

Syntax

```
FUNCTION getNotationName( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

DOM Entity Reference Methods

isNull()

Checks that the given DOMEntityRef is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( eref DOMEntityReference) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
eref	(IN)	DOMEntityReference to check.

makeNode()

Casts the DOMEntityReference to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( eref DOMEntityReference) RETURN DOMNode;
```

Parameter	IN / OUT	Description
eref	(IN)	DOMEntityReference to cast.

DOM Notation Methods

isNull()

Checks that the given DOMNotation is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( n DOMNotation) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation to check.

makeNode()

Casts the DOMNotation to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( n DOMNotation) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation to cast.

getPublicId()

Returns the public identifier of the DOMNotation.

Syntax

```
FUNCTION getPublicId( n DOMNotation) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation.

getSystemId()

Returns the system identifier of the DOMNotation.

Syntax

```
FUNCTION getSystemId( n DOMNotation) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation.

DOM Processing Instruction Methods

isNull()

Checks that the given DOMProcessingInstruction is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( pi DOMProcessingInstruction) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction to check.

makeNode()

Casts the DOMProcessingInstruction to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( pi DOMProcessingInstruction) RETURN DOMNode;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction to cast.

getData()

Returns the content data of the DOMProcessingInstruction.

Syntax

```
FUNCTION  getData( pi DOMProcessingInstruction) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.

getTarget()

Returns the target of the DOMProcessingInstruction.

Syntax

```
FUNCTION getTarget( pi DOMProcessingInstruction) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.

setData()

Sets the content data of the DOMProcessingInstruction.

Syntax

```
PROCEDURE setData( pi DOMProcessingInstruction,
                   data IN VARCHAR2);
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.
data	(IN)	New processing instruction content data.

DOM Text Methods

isNull()

Checks that the given DOMText is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( t DOMText) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText to check.

makeNode()

Casts the DOMText to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( t DOMText) RETURN DOMNode;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText to cast.

splitText()

Breaks this DOMText node into two DOMText nodes at the specified offset.

Syntax

```
FUNCTION splitText( t DOMText,
                    offset IN NUMBER)
                    RETURN DOMText;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText
offset	(IN)	Offset at which to split.

DOM Document Methods

isNull()

Checks that the given DOMDocument is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( doc DOMDocument) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument to check.

makeNode()

Casts the DOMDocument to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( doc DOMDocument) RETURN DOMNode;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument to cast.

newDOMDocument()

Returns a new DOMDocument instance.

Syntax

```
FUNCTION newDOMDocument RETURN DOMDocument;
```

freeDocument()

Frees DOMDocument object.

Syntax

```
PROCEDURE freeDocument( doc DOMDocument);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getVersion()

Returns the version information for the XML document.

Syntax

```
FUNCTION getVersion( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setVersion()

Sets version information for the XML document.

Syntax

```
PROCEDURE setVersion( doc     DOMDocument,
                      version VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
version	((N)	Version information.

getCharset()

Retrieves the character set of the XML document.

Syntax

```
FUNCTION getCharset( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setCharset()

Sets character set of the XML document.

Syntax

```
PROCEDURE setCharset( doc DOMDocument,
                      charset VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
charset	((N)	Character set.

getStandalone()

Retrieves standalone information for the XML document.

Syntax

```
FUNCTION getStandalone( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setStandalone()

Sets standalone information for the XML document.

Syntax

```
PROCEDURE setStandalone( doc DOMDocument,
                         value VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
value	((N)	Standalone information

writeToFile()

Writes XML document to a specified file. The options are given in the table below.

Syntax	Description
PROCEDURE writeToFile(doc DOMDocument, fileName VARCHAR2);	Writes XML document to a specified file using database character set.
PROCEDURE writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);	Writes XML document to a specified file using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
filename	(N)	File to write to.
charset	(IN)	Character set.

writeToBuffer()

Writes XML document to a specified buffer. The options are given in the table below.

Syntax	Description
PROCEDURE writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2);	Writes XML document to a specified buffer using database character set.
PROCEDURE writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes XML document to a specified buffer using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
buffer	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeToClob()

Writes XML document to a specified clob. The options are given in the table below.

Syntax	Description
PROCEDURE writeToClob(doc DOMDocument, cl IN OUT CLOB);	Writes XML document to a specified clob using database character set.
PROCEDURE writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);	Writes XML document to a specified clob using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
cl	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToFile()

Writes an external DTD to specified file. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToFile(doc DOMDocument, fileName VARCHAR2);	Writes an external DTD to specified file using the database character set.
PROCEDURE writeExternalDTDToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);	Writes an external DTD to specified file using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

Parameter	IN / OUT	Description
fileName	(N)	File to write to.
charset	(IN)	Character set.

writeExternalDTDToBuffer()

Writes an external DTD to specified buffer. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2);	Writes an external DTD to specified buffer using the database character set.
PROCEDURE writeExternalDTDToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes an external DTD to specified buffer using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
buffer	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToClob()

Writes an external DTD to specified clob. The options are given in the table below.

Syntax	Description
PROCEDURE writeExternalDTDToClob(doc DOMDocument, cl IN OUT CLOB);	Writes an external DTD to specified clob using the database character set.
PROCEDURE writeExternalDTDToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);	Writes an external DTD to specified clob using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
cl	(N)	Clob to write to.
charset	(IN)	Character set.

getDoctype()

Returns the DTD associated to the DOMDocument.

Syntax

```
FUNCTION getDoctype( doc DOMDocument) RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getImplementation()

Returns the DOMImplementation object that handles this DOMDocument.

Syntax

```
FUNCTION getImplementation( doc DOMDocument) RETURN DOMImplementation;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getDocumentElement()

Returns the child node -i.e. the document element- of the DOMDocument.

Syntax

```
FUNCTION getDocumentElement( doc DOMDocument) RETURN DOMElement;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

createElement()

Creates a DOMElement.

Syntax

```
FUNCTION createElement( doc DOMDocument,
                        tagName IN VARCHAR2)
                        RETURN DOMElement;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
tagName	(IN)	Tagname for new DOMElement.

createDocumentFragment()

Creates a DOMDocumentFragment.

Syntax

```
FUNCTION createDocumentFragment( doc DOMDocument) RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

createTextNode()

Creates a DOMText node.

Syntax

```
FUNCTION createTextNode( doc DOMDocument,
                         data IN VARCHAR2)
                         RETURN DOMText;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMText node.

createComment()

Creates a DOMComment node.

Syntax

```
FUNCTION createComment( doc DOMDocument,
                        data IN VARCHAR2)
                        RETURN DOMComment;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMComment node.

createCDATASection()

Creates a DOMCDATASection node.

Syntax

```
FUNCTION createCDATASection( doc DOMDocument,
                             data IN VARCHAR2)
                             RETURN DOMCDATASection;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMCDATASection node.

createProcessingInstruction()

Creates a DOMProcessingInstruction node.

Syntax

```
FUNCTION createProcessingInstruction( doc DOMDocument,
                                      target IN VARCHAR2,
                                      data IN VARCHAR2)
                                      RETURN DOMProcessingInstruction;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
target	(IN)	Target of the new processing instruction.
data	(IN)	Content data of the new processing instruction.

createAttribute()

Creates a DOMAttr node.

Syntax

```
FUNCTION createAttribute( doc DOMDocument,
                          name IN VARCHAR2)
                          RETURN DOMAttr;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
name	(IN)	New attribute name.

createEntityReference()

Creates a DOMEntityReference node.

Syntax

```
FUNCTION createEntityReference( doc DOMDocument,
                                name IN VARCHAR2)
                                RETURN DOMEntityReference;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
name	(IN)	New entity reference name.

getElementsByTagName()

Returns a DOMNodeList of all the elements with a given tagname.

Syntax

```
FUNCTION getElementsByTagName( doc DOMDocument,
                               tagname IN VARCHAR2)
                               RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
tagname	(IN)	Name of the tag to match on.

85

DBMS_XMLGEN

DBMS_XMLGEN converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a CLOB.

This package is similar to the DBMS_XMLQUERY package, except that it is written in C and compiled into the kernel. This package can only be run on the database.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB* for more information on XML support.
>
> *Oracle9i XML Database Developer's Guide - XDB*, for more information on XML support and for an example of how to use DBMS_XMLGEN.

This chapter details the following:

Description of DMS_XMLGEN

DBMS_XMLGEN converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a CLOB.

This package is similar to the DBMS_XMLQUERY package, except that it is written in C and compiled into the kernel. This package can only be run in the database.

Functions and Procedures of DBMS_XMLGEN

Table 85-1: Summary of Functions and Procedures of DBMS_XMLGEN

Function/Procedure	Description
newContext() on page 85-3	Creates a new context handle.
setRowTag() on page 85-3	Sets the name of the element enclosing each row of the result. The default tag is ROW.
setRowSetTag () on page 85-4	Sets the name of the element enclosing the entire result. The default tag is ROWSET.
getXML() on page 85-4	Gets the XML document.
getNumRowsProcessed() on page 85-5	Gets the number of SQL rows that were processed in the last call to getXML.
setMaxRows() on page 85-6	Sets the maximum number of rows to be fetched each time.
setSkipRows() on page 85-6	Sets the number of rows to skip every time before generating the XML. The default is 0.
setConvertSpecialChars() on page 85-7	Sets whether special characters such as $, which are non-XML characters, should be converted or not to their escaped representation. The default is to perform the conversion.
convert() on page 85-7	Converts the XML into the escaped or unescaped XML equivalent.
useItemTagsForColl() on page 85-8	Forces the use of the collection column name appended with the tag _ITEM for collection elements. The default is to set the underlying object type name for the base element of the collection.
restartQUERY() on page 85-8	Restarts the query to start fetching from the beginning.
closeContext() on page 85-9	Closes the context and releases all resources.

newContext()

Generates and returns a new context handle; this context handle is used in `getXML()` and other functions to get XML back from the result. The available options are given in the table below.

Syntax	Description
DBMS_XMLGEN.newContext (query IN VARCHAR2) RETURN ctxHandle;	Generates a new context handle from a query.
DBMS_XMLGEN.newContext (queryString IN SYS_REFCURSOR) RETURN ctxHandle;	Generates a new context handle from a query string in the form of a PL/SQL ref cursor

Parameter	IN / OUT	Description
query	(IN)	The query, in the form of a `VARCHAR`, the result of which must be converted to XML
queryString	(IN)	The query string in the form of a PL/SQL ref cursor, the result of which must be converted to XML.

setRowTag()

Sets the name of the element separating all the rows. The default name is `ROW`. User can set this to `NULL` to suppress the `ROW` element itself. However, an error is produced if both the row and the rowset are `NULL` and there is more than one column or row in the output; this is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

Syntax

```
DBMS_XMLGEN.setRowTag (
   ctx     IN ctxHandle,
   rowTag  IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.

Parameter	IN / OUT	Description
rowTag	(IN)	The name of the ROW element. Passing NULL indicates that you do not want the ROW element present.

setRowSetTag ()

Sets the name of the root element of the document. The default name is ROWSET. User can set the rowSetTag NULL to suppress the printing of this element. However, an error is produced if both the row and the rowset are NULL and there is more than one column or row in the output; this is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

Syntax

```
DBMS_XMLGEN.setRowSetTag (
   ctx        IN ctxHandle,
   rowSetTag  IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the newContext call.
rowSetTag	(IN)	The name of the document element. Passing NULL indicates that you do not want the ROWSET element present.

getXML()

Gets the XML document. When the rows indicated by the setSkipRows() call are skipped, the maximum number of rows as specified by the setMaxRows() call (or the entire result if not specified) is fetched and converted to XML. Use the getNumRowsProcessed() to check if any rows were retrieved. The available options are given in the table below.

Syntax	Description
FUNCTION DBMS_XMLGEN.getXML (ctx IN ctxHandle, clobval IN OUT NCOPY clob, dtdOrSchema IN number := NONE) RETURN boolean;	This procedure gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in. Use this version of getXML() to avoid any extra CLOB copies and to reuse the same CLOB for subsequent calls. Because of the CLOB reuse, this getXML() call is potentially more efficient.

Syntax	Description
FUNCTION DBMS_XMLGEN.getXML (ctx IN ctxHandle, dtdOrSchema IN number := NONE) RETURN clob;	Generates the XML document and returns it as a temporary CLOB. The temporary CLOB obtained from this function must be freed using the DBMS_LOB.FREETEMPORARY call.
FUNCTION DBMS_XMLGEN.getXML (sqlQuery IN VARCHAR2, dtdOrSchema IN number := NONE) RETURN clob;	Converts the results from the SQL query string to XML format, and returns the XML as a temporary CLOB. This temporary CLOB must be subsequently freed using the DBMS_LOB.FREETEMPORARY call.
FUNCTION DBMS_XMLGEN.getXMLType (ctx IN ctxhandle, dtdOrSchema IN number := NONE) RETURN sys.XMLType;	Generates the XML document and returns it as a sys.XMLType. XMLType operations can be performed on the results, including ExistsNode and Extract. This also provides a way of obtaining the results as a string by using the getStringVal() function, if the result size is less than 4K.
FUNCTION DBMS_XMLGEN.getXMLType (sqlQuery IN VARCHAR2, dtdOrSchema IN number := NONE) RETURN sys.XMLType	Converts the results from the SQL query string to XML format, and returns the XML as a sys.XMLType. XMLType operations can be performed on the results, including ExistsNode and Extract. This also provides a way of obtaining the results as a string by using the getStringVal() function, if the result size is less than 4K.

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the newContext call.
clobval	(IN/OUT)	The clob to which the XML document is appended.
sqlQuery	(IN)	The SQL query string.
dtdOrSchema	(IN)	The Boolean to indicate generation of either a DTD or a schema. NONE is the only option currently supported.

getNumRowsProcessed()

Retrieves the number of SQL rows processed when generating the XML using the getXML call; this count does not include the number of rows skipped before generating the XML. Used to determine the terminating condition if calling getXML() in a loop. Note that getXML() always generates an XML document, even if there are no rows present.

Syntax

```
DBMS_XMLGEN.getNumRowsProcessed (
   ctx  IN ctxHandle)
RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.

setMaxRows()

Sets the maximum number of rows to fetch from the SQL query result for every invokation of the `getXML` call. Used when generating paginated results. For example, when generating a page of XML or HTML data, restrict the number of rows converted to XML or HTML by setting the `maxRows` parameter.

Syntax

```
DBMS_XMLGEN.setMaxRows (
   ctx      IN ctxHandle,
   maxRows  IN NUMBER);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the query executed.
maxRows	(IN)	The maximum number of rows to get per call to `getXML`.

setSkipRows()

Skips a given number of rows before generating the XML output for every call to the `getXML` routine. Used when generating paginated results for stateless Web pages using this utility. For example, when generating the first page of XML or HTML data, set `skipRows` to zero. For the next set, set the `skipRows` to the number of rows obtained in the first case. See `getNumRowsProcessed()`.

Syntax

```
DBMS_XMLGEN.setSkipRows (
   ctx       IN ctxHandle,
   skipRows  IN NUMBER);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the query executed.
skipRows	(IN)	The number of rows to skip per call to `getXML`.

setConvertSpecialChars()

Sets whether or not special characters in the XML data must be converted into their escaped XML equivalent. For example, the < sign is converted to `<`. The default is to perform conversions. Improves performance of XML processing when the input data cannot contain any special characters such as <, >, ", ', which must be escaped. It is expensive to scan the character data to replace the special characters, particularly if it involves a lot of data. Syntax

```
DBMS_XMLGEN.setConvertSpecialChars (
   ctx   IN ctxHandle,
   conv  IN boolean);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.
conv	(IN)	`TRUE` indicates that conversion is needed.

convert()

Converts the XML data into the escaped or unescaped XML equivalent; returns XML CLOB data in encoded or decoded format. Escapes the XML data if the `ENTITY_ENCODE` is specified. For example, the escaped form of the character < is `<`. Unescaping is the reverse transformation. The available options are given in the table below.

Syntax	Description
DBMS_XMLGEN.convert (xmlData IN VARCHAR2, flag IN NUMBER := ENTITY_ENCODE) RETURN VARCHAR2;	Uses xmlData in string form (VARCHAR2).

Syntax	Description
DBMS_XMLGEN.convert (xmlData IN CLOB, flag IN NUMBER := ENTITY_ENCODE) RETURN CLOB;	Uses xmlData in Clob form.

Parameter	IN / OUT	Description
xmlData	(IN)	The XML CLOB data to be encoded or decoded.
flag	(IN)	The flag setting; ENTITY_ENCODE (default) for encode, and ENTITY_DECODE for decode.

useItemTagsForColl()

Overrides the default name of the collection elements. The default name for collection elements is the type name itself. Using this function, you can override the default to use the name of the column with the _ITEM tag appended to it. If there is a collection of NUMBER, the default tag name for the collection elements is NUMBER. Using this procedure, the user can override this behavior and generate the collection column name with the _ITEM tag appended to it.

Syntax

```
DBMS_XMLGEN.useItemTagsForColl (
   ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle.

restartQUERY()

Restarts the query and generates the XML from the first row. Can be used to start executing the query again, without having to create a new context.

Syntax

```
DBMS_XMLGEN.restartQUERY (ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the current query.

closeContext()

Closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers. After this call, the handle cannot be used for a subsequent DBMS_XMLGEN function call.

Syntax

```
DBMS_XMLGEN.closeContext ( ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle to close.

86

DBMS_XMLPARSER

Using DBMS_XMLPARSER, you can access the contents and structure of XML documents.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB* for more information

This chapter details the following:

Description of DBMS_XMLPARSER

The Extensible Markup Language (XML) describes a class of data objects called XML documents. It partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of the Standard Generalized Markup Language (SGML). By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This PL/SQL implementation of the XML processor (or parser) followed the W3C XML specification (rev. REC-xml-19980210) and included the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The following is the default behavior for this PL/SQL XML parser:

- A parse tree which can be accessed by DOM APIs is built
- The parser is validating if a DTD is found, otherwise, it is non-validating
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Functions and Procedures of DBMS_XMLPARSER

Table 86-1: Summary of Functions and Procedures of DBMS_XMLPARSER

Subprogram	Description
parse() on page 86-3	Parses XML stored in the given url/file.
newParser() on page 86-4	Returns a new parser instance
parseBuffer() on page 86-4	Parses XML stored in the given buffer
parseClob() on page 86-4	Parses XML stored in the given clob
parseDTD() on page 86-5	Parses DTD stored in the given url/file

Table 86-1: Summary of Functions and Procedures of DBMS_XMLPARSER

Subprogram	Description
parseDTDBuffer() on page 86-5	Parses DTD stored in the given buffer
parseDTDClob() on page 86-6	Parses DTD stored in the given clob
setBaseDir() on page 86-6	Sets base directory used to resolve relative URLs.
showWarnings() on page 86-6	Turns warnings on or off.
setErrorLog() on page 86-7	Sets errors to be sent to the specified file
setPreserveWhitespace() on page 86-7	Sets white space preserve mode
setValidationMode() on page 86-8	Sets validation mode.
getValidationMode() on page 86-8	Returns validation mode.
setDoctype() on page 86-8	Sets DTD.
getDoctype() on page 86-9	Gets DTD Parser.
getDocument() on page 86-9	Gets DOM document.
freeParser() on page 86-9	Frees a parser object.
getReleaseVersion() on page 86-10	Returns the release version of Oracle XML Parser for PL/SQL.

parse()

Parses XML stored in the given url/file. An application error is raised if parsing fails. The options are described in the following table.

Syntax	Description
FUNCTION parse(url VARCHAR2) RETURN DOMDocument;	Returns the built DOM Document. This is meant to be used when the default parser behavior is acceptable and just a url/file needs to be parsed.
PROCEDURE parse(p Parser, url VARCHAR2);	Any changes to the default parser behavior should be effected before calling this procedure.

Parameter	IN / OUT	Description
url	(IN)	Complete path of the url/file to be parsed.
p	(IN)	Parser instance.

newParser()

Returns a new parser instance. This function must be called before the default behavior of Parser can be changed and if other parse methods need to be used.

Syntax

```
FUNCTION newParser RETURN Parser;
```

parseBuffer()

Parses XML stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseBuffer( p   Parser,
                       doc VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

parseClob()

Parses XML stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseClob( p   Parser,
                     doc CLOB);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

parseDTD()

Parses the DTD stored in the given url/file. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTD( p     Parser,
                    url   VARCHAR2,
                    root  VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
url	(IN)	Complete path of the url/file to be parsed.
p	(IN)	Parser instance.

parseDTDBuffer()

Parses the DTD stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTDBuffer( p    Parser,
                          dtd  VARCHAR2,
                          root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD buffer to parse.

Parameter	IN / OUT	Description
root	(IN)	Name of the root element.

parseDTDClob()

Parses the DTD stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTDClob( p    Parser,
                        dtd  CLOB,
                        root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD Clob to parse.
root	(IN)	Name of the root element.

setBaseDir()

Sets base directory used to resolve relative URLs. An application error is raised if parsing fails.

Syntax

```
PROCEDURE setBaseDir( p   Parser,
                      dir VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dir	(IN)	Directory used as a base directory.

showWarnings()

Turns warnings on or off.

Syntax

```
PROCEDURE showWarnings( p   Parser,
                        yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - show warnings, FALSE - don't show warnings.

setErrorLog()

Sets errors to be sent to the specified file

Syntax

```
PROCEDURE setErrorLog( p        Parser,
                       fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
fileName	(IN)	Complete path of the file to use as the error log.

setPreserveWhitespace()

Sets whitespace preserving mode.

Syntax

```
PROCEDURE setPreserveWhitespace( p   Parser,
                                 yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - preserve, FALSE - don't preserve.

setValidationMode()

Sets validation mode.

Syntax

```
PROCEDURE setValidationMode( p   Parser,
                             yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - validate, FALSE - don't validate.

getValidationMode()

Retrieves validation mode; TRUE for validating, FALSE otherwise.

Syntax

```
FUNCTION getValidationMode( p Parser)
                            RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

setDoctype()

Sets a DTD to be used by the parser for validation. This call should be made before the document is parsed.

Syntax

```
PROCEDURE setDoctype( p   Parser,
                      dtd DOMDocumentType);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

Parameter	IN / OUT	Description
dtd	(IN)	DTD to set.

getDoctype()

Returns the parsed DTD; this function MUST be called only after a DTD is parsed.

Syntax

```
FUNCTION getDoctype( p Parser)
                    RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

getDocument()

Returns the root of the DOM tree document built by the parser; this function MUST be called only after a document is parsed.

Syntax

```
FUNCTION getDocument( p Parser)
                     RETURN DOMDocument;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

freeParser()

Frees a parser object.

Syntax

```
PROCEDURE freeParser( p Parser);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

getReleaseVersion()

Returns the release version of the Oracle XML parser for PL/SQL.

Syntax

```
PROCEDURE getReleaseVersion RETURN VARCHAR2;
```

87

DBMS_XMLQUERY

DBMS_XMLGEN is a built-in package in C. In general, use DBMS_XMLGEN instead of DBMS_XMLQUERY wherever possible. DBMS_XMLQUERY provides database-to-XMLType functionality.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle DB for XML* for more information

This chapter details the following:

- Types of DBMS_XMLQuery
- Constants of DBMS_XMLQuery
- Functions and Procedures of DBMS_XMLQuery

Description of DBMS_XMLQuery

This API provides DB_to_XML type functionality.

Types of DBMS_XMLQuery

Table 87-1: Types of DBMS_XMLQuery

Type	Description
ctxType	The type of the query context handle. This is the return type of newContext().

Constants of DBMS_XMLQuery

Table 87-2: Constants of DBMS_XMLQuery

Constant	Description
DB_ENCODING	Used to signal that the DB character encoding is to be used.
DEFAULT_ROWSETTAG	The tag name for the element enclosing the XML generated from the result set (that is, for most cases the root node tag name) -- ROWSET.
DEFAULT_ERRORTAG	The default tag to enclose raised errors -- ERROR.
DEFAULT_ROWIDATTR	The default name for the cardinality attribute of XML elements corresponding to db. records. -- NUM
DEFAULT_ROWTAG	The default tag name for the element corresponding to db. records. -- ROW
DEFAULT_DATE_FORMAT	Default date mask. -- 'MM/dd/yyyy HH:mm:ss'
ALL_ROWS	The ALL_ROWS parameter is to indicate that all rows are needed in the output.
NONE	Used to specifies that the output should not contain any XML metadata (for example, no DTD or Schema).
DTD	Used to specify that the generation of the DTD is desired.
SCHEMA	Used to specify that the generation of the XML SCHEMA is desired.
LOWER_CASE	Use lower cased tag names.
UPPER_CASE	Use upper case tag names.

Functions and Procedures of DBMS_XMLQuery

Table 87-3: Summary of Functions and Procedures of DBMS_XMLQuery

Functions/Procedures	Description
newContext() on page 87-4	Creates a query context and it returns the context handle.
closeContext() on page 87-5	Closes / deallocates a particular query context.
setRowsetTag() on page 87-5	Sets the tag to be used to enclose the XML dataset.
setRowTag() on page 87-5	Sets the tag to be used to enclose the XML element corresponding to a db.
setErrorTag() on page 87-6	Sets the tag to be used to enclose the XML error docs.
setRowIdAttrName() on page 87-6	Sets the name of the id attribute of the row enclosing tag.
setRowIdAttrValue() on page 87-6	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
setCollIdAttrName() on page 87-7	Sets the name of the id attribute of the collection element's separator tag.
useNullAttributeIndicator() on page 87-7	Specifies weather to use an XML attribute to indicate NULLness.
useTypeForCollElemTag() on page 87-8	Tells the XSU to use the collection element's type name as the collection element tag name.
setTagCase() on page 87-8	Specified the case of the generated XML tags.
setDateFormat() on page 87-8	Sets the format of the generated dates in the XML doc.
setMaxRows() on page 87-9	Sets the max number of rows to be converted to XML.
setSkipRows() on page 87-9	Sets the number of rows to skip.
setStylesheetHeader() on page 87-10	Sets the stylesheet header.
setXSLT() on page 87-10	Registers a stylesheet to be applied to generated XML.
setXSLTParam() on page 87-11	Sets the value of a top-level stylesheet parameter.
removeXSLTParam() on page 87-11	Removes a particular top-level stylesheet parameter.
setBindValue() on page 87-12	Sets a value for a particular bind name.
setMetaHeader() on page 87-12	Sets the XML meta header.
setDataHeader() on page 87-12	Sets the XML data header.

Table 87-3: Summary of Functions and Procedures of DBMS_XMLQuery

Functions/Procedures	Description
setEncodingTag() on page 87-13	Sets the encoding processing instruction in the XML document.
setRaiseException() on page 87-13	Tells the XSU to throw the raised exceptions.
setRaiseNoRowsException() on page 87-14	Tells the XSU to throw or not to throw an `OracleXMLNoRowsException` in the case when for one reason or another, the XML doc generated is empty.
setSQLToXMLNameEscaping() on page 87-14	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
propagateOriginalException() on page 87-15	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.
getExceptionContent() on page 87-15	Via its arguments, this method returns the thrown exception's error code and error message.
getDTD() on page 87-15	Generates the DTD.
getNumRowsProcessed() on page 87-16	Return the number of rows processed for the query.
getVersion() on page 87-16	Prints the version of the XSU in use.
getXML() on page 87-17	Generates the XML document.

newContext()

Creates a query context and it returns the context handle. The options are described in the following table.

Syntax	Description
FUNCTION newContext(sqlQuery IN VARCHAR2) RETURN ctxType	Creates a query context from a string.
FUNCTION newContext(sqlQuery IN CLOB) RETURN ctxType	Creates a query context from a CLOB.

Parameter	IN / OUT	Description
sqlQuery	(IN)	SQL query, the results of which to convert to XML.

closeContext()

Closes / deallocates a particular query context

Syntax

```
PROCEDURE closeContext( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setRowsetTag()

Sets the tag to be used to enclose the XML dataset.

Syntax

```
PROCEDURE setRowsetTag(ctxHdl IN ctxType, tag IN VARCHAR2)
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowTag()

Sets the tag to be used to enclose the XML element corresponding to a db. record.

Syntax

```
PROCEDURE setRowTag(ctxHdl IN ctxType, tag IN VARCHAR2)
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setErrorTag()

Sets the tag to be used to enclose the XML error docs.

Syntax

```
PROCEDURE setErrorTag( ctxHdl IN ctxType,
                       tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowIdAttrName()

Sets the name of the id attribute of the row enclosing tag. Passing null or an empty string for the tag results the row id attribute to be omitted.

Syntax

```
PROCEDURE setRowIdAttrName( ctxHdl IN ctxType,
                            attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowIdAttrValue()

Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag. Passing null or an empty string for the colName results the row id attribute being assigned the row count value (that is, 0, 1, 2 and so on).

Syntax

```
PROCEDURE setRowIdAttrValue( ctxHdl IN ctxType,
                             colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column whose value is to be assigned to the row id attribute.

setCollIdAttrName()

Sets the name of the id attribute of the collection element's separator tag.

Syntax

```
PROCEDURE setCollIdAttrName( ctxHdl IN ctxType,
                             attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
attrName	(IN)	AttributeName.

useNullAttributeIndicator()

Specified weather to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document.

Syntax

```
PROCEDURE useNullAttributeIndicator( ctxHdl IN ctxType,
                                     flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Use attribute to indicate NULL?

useTypeForCollElemTag()

By default the tag name for elements of a collection is the collection's tag name followed by "_item". This method, when called with argument of TRUE, tells the XSU to use the collection element's type name as the collection element tag name.

Syntax

```
PROCEDURE useTypeForCollElemTag( ctxHdl IN ctxType,
                                 flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Turn on use of the type name?.

setTagCase()

Specified the case of the generated XML tags.

Syntax

```
PROCEDURE setTagCase( ctxHdl IN ctxType,
                      tCase IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tCase	(IN)	The tag's case; 0=asAre, 1=lower, 2=upper.

setDateFormat()

Sets the format of the generated dates in the XML doc. The syntax of the date format pattern (that is, the date mask), should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to null or an empty string, results the use of the default mask -- DEFAULT_DATE_FORMAT.

Syntax

```
PROCEDURE setDateFormat( ctxHdl IN ctxType,
                         mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
mask	(IN)	The date mask.

setMaxRows()

Sets the max number of rows to be converted to XML. By default there is no max set.

Syntax

```
PROCEDURE setMaxRows ( ctxHdl IN ctxType,
                       rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to generate.

setSkipRows()

Sets the number of rows to skip. By default 0 rows are skipped.

Syntax

```
PROCEDURE setSkipRows( ctxHdl IN ctxType,
                       rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to skip.

setStylesheetHeader()

Sets the stylesheet header (that is, stylesheet processing instructions) in the generated XML doc. Note: Passing null for the uri argument will unset the stylesheet header and the stylesheet type.

Syntax

```
PROCEDURE setStylesheetHeader( ctxHdl IN ctxType,
                               uri IN VARCHAR2,
                               type IN VARCHAR2 := 'text/xsl');
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	Stylesheet URI.
type	(IN)	Stylesheet type; defaults to "`text/xsl`".

setXSLT()

Registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it gets replaced by the new one. The options are described in the following table.

Syntax	Description
PROCEDURE setXSLT(ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);	To un-register the stylesheet pass in a null for the uri.
PROCEDURE setXSLT(ctxHdl IN ctxType, stylesheet CLOB, ref IN VARCHAR2 := null);	To un-register the stylesheet pass in a null or an empty string for the stylesheet.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

Parameter	IN / OUT	Description
uri	(IN)	Stylesheet URI.
stylesheet	(IN)	Stylesheet.
ref	(IN)	URL to include, import and external entities.

setXSLTParam()

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression (note that string literal values would therefore have to be explicitly quoted). NOTE: if no stylesheet is registered, this method is a no op.

Syntax

```
PROCEDURE setXSLTParam( ctxHdl IN ctxType,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.
value	(IN)	Value to be assigned to the stylesheet parameter.

removeXSLTParam()

Removes the value of a top-level stylesheet parameter. NOTE: if no stylesheet is registered, this method is a no op.

Syntax

```
PROCEDURE removeXSLTParam( ctxHdl IN ctxType,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.

setBindValue()

Sets a value for a particular bind name.

Syntax

```
PROCEDURE setBindValue( ctxHdl IN ctxType,
                        bindName IN VARCHAR2,
                        bindValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
bindName	(IN)	Bind name.
bindValue	(IN)	Bind value.

setMetaHeader()

Sets the XML meta header. When set, the header is inserted at the beginning of the metadata part (DTD or XMLSchema) of each XML document generated by this object. Note that the last meta header specified is the one that is used; furthermore, passing in null for the header, parameter unsets the meta header.

Syntax

```
PROCEDURE setMetaHeader( ctxHdl IN ctxType,
                         header IN CLOB := null);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
Header	(IN)	Header.

setDataHeader()

Sets the XML data header. The data header is an XML entity which is appended at the beginning of the query-generated XML entity (i.e. rowset). The two entities are enclosed by the tag specified via the docTag argument. Note that the last data header specified is the one that is used; furthermore, passing in null for the header, parameter unsets the data header.

Syntax

```
PROCEDURE setDataHeader( ctxHdl IN ctxType,
                         header IN CLOB := null,
                         tag IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
header	(IN)	Header.
tag	(IN)	Tag used to enclose the data header and the rowset.

setEncodingTag()

Sets the encoding processing instruction in the XML document.

Syntax

```
PROCEDURE setEncodingTag( ctxHdl IN ctxType,
                          enc IN VARCHAR2 := DB_ENCODING);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
enc	(IN)	The encoding to use.

setRaiseException()

Tells the XSU to throw the raised exceptions. If this call isn't made or if false is passed to the flag argument, the XSU catches the SQL exceptions and generates an XML doc out of the exception's message.

Syntax

```
PROCEDURE setRaiseException( ctxHdl IN ctxType,
                             flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Throw raised exceptions? TRUE for yes, otherwise FALSE.

setRaiseNoRowsException()

Tells the XSU whether to throw an OracleXMLNoRowsException in the case when for one reason or another, the XML doc generated is empty. By default, the exception is not thrown.

Syntax

```
PROCEDURE setRaiseNoRowsException( ctxHdl IN ctxType,
                                   flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Throw OracleXMLNoRowsException if no data? TRUE for yes, otherwise FALSE.

setSQLToXMLNameEscaping()

This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
PROCEDURE setSQLToXMLNameEscaping( ctxHdl IN ctxType,
                                   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping? TRUE for yes, otherwise FALSE.

propagateOriginalException()

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.

Syntax

```
PROCEDURE propagateOriginalException( c txHdl IN ctxType,
                                      flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Propagate original exception? TRUE for yes, otherwise FALSE.

getExceptionContent()

Via its arguments, this method returns the thrown exception's error code and error message (that is, SQL error code). This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent( ctxHdl IN ctxType,
                               errNo OUT NUMBER,
                               errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN	Error message.

getDTD()

Generates and returns the DTD based on the SQL query used to initialize the context. The options are described in the following table.

Syntax	Description
FUNCTION getDTD(ctxHdl IN ctxType, withVer IN BOOLEAN := false) RETURN CLOB;	Function that generates the DTD based on the SQL query used to initialize the context.
PROCEDURE getDTD(ctx IN ctxType, xDoc IN CLOB, withVer IN BOOLEAN := false);	Procedure that generates the DTD based on the SQL query used to initialize the context and xDOC in CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
withVer	(IN)	Generate the version info? TRUE for yes, otherwise FALSE.
xDoc	(IN)	Clob into which to write the generated XML doc.

getNumRowsProcessed()

Return the number of rows processed for the query.

Syntax

```
FUNCTION getNumRowsProcessed( ctx IN ctxType) RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

getVersion()

Prints the version of the XSU in use.

Syntax

```
PROCEDURE getVersion();
```

getXML()

Creates the new context, executes the query, gets the XML back and closes the context. This is a convenience function. The context doesn't have to be explicitly opened or closed. The options are described in the following table.

Syntax	Description
FUNCTION getXML(sqlQuery IN VARCHAR2, metaType IN NUMBER := NONE) RETURN CLOB;	This function uses the sqlQuery in string form.
FUNCTION getXML(sqlQuery IN CLOB, metaType IN NUMBER := NONE) RETURN CLOB;	This function uses the sqlQuery in clob form.
FUNCTION getXML(ctxHdl IN ctxType, metaType IN NUMBER := NONE); RETURN CLOB	This function generates the XML doc. based on the SQL query used to initialize the context.
PROCEDURE getXML(ctxHdl IN ctxType, xDoc IN CLOB, metaType IN NUMBER := NONE);	This procedure generates the XML doc. based on the SQL query used to initialize the context.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
sqlQuery	(IN)	SQLQuery.
metaType	(IN)	XML metadata type (NONE, DTD, or SCHEMA).
sDoc	(IN)	Clob into which to write the generated XML doc.

88

DBMS_XMLSAVE

DBMS_XMLSAVE provides XML to database-type functionality.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle DB for XML* for more information

This chapter details the following:

Description of DBMS_XMLSave

This API provides XML_to_DB type functionality.

Types of DBMS_XMLSave

Table 88-1: Types of DBMS_XMLSave

Type	Description
ctxType	The type of the query context handle. The type of the query context handle. This the return type of newContext().

Constants of DBMS_XMLSave

Table 88-2: Constants of DBMS_XMLSave

Constant	Description
DEFAULT_ROWTAG	The default tag name for the element corresponding to db. records. -- ROW
DEFAULT_DATE_FORMAT	Default date mask. -- 'MM/dd/yyyy HH:mm:ss'
MATCH_CASE	Used to specify that when mapping XML elements to DB. entities the XSU should be case sensitive.
IGNORE_CASE	Used to specify that when mapping XML elements to DB. entities the XSU should be case insensitive.

Functions and Procedures of DBMS_XMLSave

Table 88-3: Summary of Functions and Procedures of DBMS_XMLSave

Functions/Procedures	Description
newContext() on page 88-3	Creates a save context, and returns the context handle.
closeContext() on page 88-4	It closes/deallocates a particular save context.
setRowTag() on page 88-4	Names the tag used in the XML doc., to enclose the XML elements corresponding to db.
setIgnoreCase() on page 88-4	The XSU does mapping of XML elements to db.

Table 88-3: Summary of Functions and Procedures of DBMS_XMLSave

Functions/Procedures	Description
setDateFormat() on page 88-5	Describes to the XSU the format of the dates in the XML document.
setBatchSize() on page 88-5	Changes the batch size used during DML operations.
setCommitBatch() on page 88-6	Sets the commit batch size.
setSQLToXMLNameEscaping() on page 88-6	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
setUpdateColumn() on page 88-7	Adds a column to the "update column list".
clearUpdateColumnList() on page 88-7	Clears the update column list.
setPreserveWhitespace() on page 88-7	Tells the XSU whether to preserve whitespace or not.
setKeyColumn() on page 88-8	This methods adds a column to the "key column list".
clearKeyColumnList() on page 88-8	Clears the key column list.
setXSLT() on page 88-8	Registers a XSL transform to be applied to the XML to be saved.
setXSLTParam() on page 88-9	Sets the value of a top-level stylesheet parameter.
removeXSLTParam() on page 88-10	Removes the value of a top-level stylesheet parameter
insertXML() on page 88-10	Inserts the XML document into the table specified at the context creation time.
updateXML() on page 88-11	Updates the table given the XML document.
deleteXML() on page 88-11	Deletes records specified by data from the XML document, from the table specified at the context creation time.
propagateOriginalException() on page 88-12	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.
getExceptionContent() on page 88-12	Via its arguments, this method returns the thrown exception's error code and error message.

newContext()

Creates a save context, and returns the context handle.

Syntax

```
FUNCTION newContext(t argetTable IN VARCHAR2) RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML doc.

closeContext()

Closes / deallocates a particular save context

Syntax

```
PROCEDURE closeContext(ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setRowTag()

Names the tag used in the XML doc., to enclose the XML elements corresponding to db. records.

Syntax

```
PROCEDURE setRowTag( ctxHdl IN ctxType,
                     tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setIgnoreCase()

The XSU does mapping of XML elements to db columns / attributes based on the element names (XML tags). This function tells the XSU to do this match case insensitive.

Syntax

```
PROCEDURE setIgnoreCase( ctxHdl IN ctxType,
                         flag IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Ignore tag case in the XML doc? 0=FALSE, 1=TRUE.

setDateFormat()

Describes to the XSU the format of the dates in the XML document. The syntax of the date format pattern (that is, the date mask), should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to null or an empty string, results the use of the default mask -- OracleXMLCore.DATE_FORMAT.

Syntax

```
PROCEDURE setDateFormat( ctxHdl IN ctxType,
                         mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
mask	(IN)	The date mask.

setBatchSize()

Changes the batch size used during DML operations. When performing inserts, updates or deletes, it is better to batch the operations so that they get executed in one shot rather than as separate statements. The flip side is that more memory is needed to buffer all the bind values. Note that when batching is used, a commit occurs only after a batch is executed. So if one of the statement inside a batch fails, the whole batch is rolled back. This is a small price to pay considering the performance gain; nevertheless, if this behavior is unacceptable, then set the batch size to 1.

Syntax

```
PROCEDURE setBatchSize( ctxHdl IN ctxType,
```

```
                                batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Batch size.

setCommitBatch()

Sets the commit batch size. The commit batch size refers to the number or records inserted after which a commit should follow. Note that if commitBatch is < 1 or the session is in "auto-commit" mode then the XSU does not make any explicit commit's. By default the commit-batch size is 0.

Syntax

```
PROCEDURE setCommitBatch( ctxHdl IN ctxType,
                          batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Commit batch size.

setSQLToXMLNameEscaping()

Turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
PROCEDURE setSQLToXMLNameEscaping( ctxHdl IN ctxType,
                                   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping?

setUpdateColumn()

Adds a column to the "update column list". In case of insert, the default is to insert values to all the columns in the table; on the other hand, in case of updates, the default is to only update the columns corresponding to the tags present in the ROW element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

Syntax

```
PROCEDURE setUpdateColumn( ctxHdl IN ctxType,
                           colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the update column list.

clearUpdateColumnList()

Clears the update column list.

Syntax

```
PROCEDURE clearUpdateColumnList( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setPreserveWhitespace()

Tells the XSU whether or not to preserve whitespace.

Syntax

```
PROCEDURE setPreserveWhitespace( ctxHdl IN ctxType,
                                 flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Should XSU preserve whitespace?

setKeyColumn()

This methods adds a column to the "key column list". In case of update or delete, it is the columns in the key column list that make up the where clause of the update / delete statement. The key columns list must be specified before updates can be done; yet, it is only optional for delete operations.

Syntax

```
PROCEDURE setKeyColumn( ctxHdl IN ctxType,
                        colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list.

clearKeyColumnList()

Clears the key column list.

Syntax

```
PROCEDURE clearKeyColumnList( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setXSLT()

Registers an XSL transform to be applied to the XML to be saved. If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet, pass in null for the URI. The options are described in the following table.

Syntax	Description
PROCEDURE setXSLT(ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);	Passes in the stylesheet through a URI.
PROCEDURE setXSLT(ctxHdl IN ctxType, stylesheet IN CLOB, ref IN VARCHAR2 := null);	Passes in the stylesheet through a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	URI to the stylesheet to register.
ref	(IN)	URL for include, import, and external entities.
stylesheet	(IN)	CLOB containing the stylesheet to register.

setXSLTParam()

Sets the value of a top-level stylesheet parameter. The parameter is expected to be a valid XPath expression (not that string literal values would therefore have to be explicitly quoted).

Syntax

```
PROCEDURE setXSLTParam( ctxHdl IN ctxType,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.
value	(IN)	Parameter value as an XPath expression

removeXSLTParam()

Removes the value of a top-level stylesheet parameter.

Syntax

```
PROCEDURE removeXSLTParam( ctxHdl IN ctxType,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.

insertXML()

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted. The options are described in the following table.

Syntax	Description
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;	Passes in the xDoc parameter as a VARCHAR2.
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Passes in the xDoc parameter as a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

updateXML()

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;	Passes in the xDoc parameter as a VARCHAR2.
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Passes in the xDoc parameter as a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

deleteXML()

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted. The options are described in the following table.

Syntax	Description
FUNCTION deleteXML(ctxHdl IN ctxPType, xDoc IN VARCHAR2) RETURN NUMBER;	Uses a VARCHAR2 type for the xDoc parameter.
FUNCTION deleteXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Uses a CLOB type for the xDoc parameter.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

propagateOriginalException()

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.

Syntax

```
PROCEDURE propagateOriginalException( ctxHdl IN ctxType,
                                      flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Propagate the original exception? 0=FALSE, 1=TRUE.

getExceptionContent()

Through its arguments, this method returns the thrown exception's error code and error message (that is, SQL error code) This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent( ctxHdl IN ctxType,
                               errNo OUT NUMBER,
                               errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN)	Error message.

89

DBMS_XMLSchema

DBMS_XMLSchema Package provides procedures to register and delete XML schemas.

> **See Also:** *Oracle9i XML API Reference - XDK and XDB* for more information

This chapter details the following:

- Constants of DBMS_XMLSCHEMA
- Procedures and Functions of DBMS_XMLSCHEMA

Description of DBMS_XMLSCHEMA

This package is created by script dbmsxsch.sql during XDB installation. It provides procedures to register and delete XML schemas.

Constants of DBMS_XMLSCHEMA

Table 89-1: Constants of DBMS_XMLSCHEMA

Constant	Description
DELETE_RESTRICT	CONSTANT NUMBER := 1;
DELETE_INVALIDATE	CONSTANT NUMBER := 2;
DELETE_CASCADE	CONSTANT NUMBER := 3;
DELETE_CASCADE_FORCE	CONSTANT NUMBER := 4;

Procedures and Functions of DBMS_XMLSCHEMA

Table 89-2: Summary of Functions and Procedures of DBMS_XMLSCHEMA

Constant	Description
registerSchema() on page 89-2	Registers the specified schema for use by Oracle. This schema can then be used to store documents conforming to this.
registerURI() on page 89-5	Registers an XMLSchema specified by a URI name.
deleteSchema() on page 89-6	Removes the schema from Oracle XML DB.
generateBean() on page 89-6	Generates the Java bean code corresponding to a registered XML schema
compileSchema() on page 89-7	Used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state.
generateSchema() on page 89-7	Generates XML schema(s) from an oracle type name.

registerSchema()

Registers the specified schema for use by the Oracle XML DB. The available options are given in the table below.

Syntax	Description
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN VARCHAR2, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, genTables IN BOOLEAN := TRUE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers a schema specified as a VARCHAR2.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN CLOB, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FASLE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a CLOB.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN BFILE, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a BFILE.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN SYS.XMLType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as an XMLType.

Syntax	Description
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN SYS.URIType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a URIType.

Parameter	IN / OUT	Description
schemaURL	(IN)	URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the XDB hierarchy.
schemaDoc	(IN)	a valid XML schema document
local	(IN)	Is this a local or global schema? By default, all schemas are registered as local schemas i.e. under /sys/schemas/<username/... If a schema is registered as global, it is added under /sys/schemas/PUBLIC/.... You need write privileges on the above directory to be able to register a schema as global.
genTypes	(IN)	Should the schema compiler generate object types? By default, TRUE
genbean	(IN)	Should the schema compiler generate Java beans? By default, FALSE.
genTables	(IN)	Should the schema compiler generate default tables? By default, TRUE
force	(IN)	If this parameter is set to TRUE, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is FALSE.
owner	(IN)	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

registerURI()

Registers an XMLSchema specified by a URI name.

Syntax

```
procedure registerURI(schemaURL IN varchar2,
                      schemaDocURI IN varchar2,
                      local IN BOOLEAN := TRUE,
                      genTypes IN BOOLEAN := TRUE,
                      genbean IN BOOLEAN := FALSE,
                      genTables IN BOOLEAN := TRUE,
                      force IN BOOLEAN := FALSE,
                      owner IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	A name that uniquely identifies the schema document.
schemaDocURI	(IN)	Pathname (URI) corresponding to the physical location of the schema document. The URI path could be based on HTTP, FTP, DB or XDB protocols. This function constructs a URIType instance using the URIFactory - and invokes the `registerSchema()` function.
local	(IN)	Is this a local or global schema? By default, all schemas are registered as local schemas i.e. under /sys/schemas/<username/... If a schema is regsitered as global, it is added under /sys/schemas/PUBLIC/.... User needs write privileges on the above directory to be able to register a schema as global.
genTypes	(IN)	Should the schema compiler generate object types? By default, TRUE
genbean	(IN)	Should the schema compiler generate Java beans? By default, FALSE.
genTables	(IN)	Should the schema compiler generate default tables? By default, TRUE
force	(IN)	If this parameter is set to TRUE, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is FALSE.

Parameter	IN / OUT	Description
owner	(IN)	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

deleteSchema()

Deletes the XMLSchema specified by the URL. Can result in a `ORA-31001` exception: invalid resource handle or path name.

Syntax

```
procedure deleteSchema(schemaURL IN varchar2,
                       delete_option IN pls_integer := DELETE_RESTRICT);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	URL identifying the schema to be deleted.
delete_option	(IN)	Option for deleting schema.

Options for delete_option parameter

Option	Description
DELETE_RESTRICT	Schema deletion fails if there are any tables or schemas that depend on this schema.
DELETE_INVALIDATE	Schema deletion does not fail if there are any dependencies. Instead, it simply invalidates all dependent objects.
DELETE_CASCADE	Schema deletion will also drop all default SQL types and default tables. However the deletion fails if there are any stored instances conforming to this schema.
DELETE_CASCADE_FORCE	Similar to CASCADE except that it does not check for any stored instances conforming to this schema. Also it ignores any errors.

generateBean()

This procedure can be used to generate the Java bean code corresponding to a registered XML schema. Note that there is also an option to generate the beans as

part of the registration procedure itself. Can result in a ORA-31001 exception: invalid resource handle or path name.

Syntax

```
procedure generateBean(schemaURL IN varchar2);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	Name identifying a registered XML schema.

compileSchema()

This procedure can be used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a ORA-31001 exception: invalid resource handle or path name.

Syntax

```
procedure compileSchema( schemaURL IN varchar2);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	URL identifying the schema.

generateSchema()

These functions generate XML schema(s) from an oracle type name. Can result in a ORA-31001 exception: invalid resource handle or path name. The available options are given in the table below.

Syntax	Description
function generateSchemas(schemaName IN varchar2, typeName IN varchar2, elementName IN varchar2 := NULL, schemaURL IN varchar2 := NULL, annotate IN BOOLEAN := TRUE, embedColl IN BOOLEAN := TRUE) return sys.XMLSequenceType;	Returns a collection of XMLTypes, one XMLSchema document for each database schema.
function generateSchema(schemaName IN varchar2, typeName IN varchar2, elementName IN varchar2 := NULL, recurse IN BOOLEAN := TRUE, annotate IN BOOLEAN := TRUE, embedColl IN BOOLEAN := TRUE) return sys.XMLType;	Inlines all in one schema (XMLType).

Parameter	IN / OUT	Description
schemaName	(IN)	Name of the database schema containing the type.
typeName	(IN)	Name of the oracle type.
elementName	(IN)	The name of the toplevel element in the XMLSchema defaults to typeName.
schemaURL	(IN)	Dpecifies base URL where schemas will be stored, needed by top level schema for import statement.
recurse	(IN)	Whether or not to also generate schema for all types referred to by the type specified.
annotate	(IN)	Whether or not to put the SQL annotations in the XMLSchema.
embedColl	(IN)	Should the collections be embedded in the type which refers to them, or create a complexType? Cannot be FALSE if annotations are turned on.

Catalog Views

Table 89-3: Summary of Catalog View Schemas

Schema	Description
USER_XML_SCHEMAS on page 89-9	All registered XML Schemas owned by the user.
ALL_XML_SCHEMAS on page 89-10	All registered XML Schemas usable by the current user.
DBA_XML_SCHEMAS on page 89-10	All registered XML Schemas in Oracle XML DB.
DBA_XML_TABLES on page 89-10	All XMLType tables in the system.
USER_XML_TABLES on page 89-10	All XMLType tables owned by the current user.
ALL_XML_TABLES on page 89-11	All XMLType tables usable by the current user.
DBA_XML_TAB_COLS on page 89-11	All XMLType table columns in the system.
USER_XML_TAB_COLS on page 89-11	All XMLType table columns in tables owned by the current user.
ALL_XML_TAB_COLS on page 89-12	All XMLType table columns in tables usable by the current user.
DBA_XML_VIEWS on page 89-12	All XMLType views in the system.
USER_XML_VIEWS on page 89-12	All XMlType views owned by the current user.
ALL_XML_VIEWS on page 89-13	All XMLType views usable by the current user.
DBA_XML_VIEW_COLS on page 89-13	All XMLType view columns in the system.
USER_XML_VIEW_COLS on page 89-13	All XMLType view columns in views owned by the current user.
ALL_XML_VIEW_COLS on page 89-14	All XMLType view columns in views usable by the current user.

USER_XML_SCHEMAS

Lists all schemas (local and global) belonging to the current user.

Column	Datatype	Description
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)
SCHEMA	XMLTYPE	XML Schema document

ALL_XML_SCHEMAS

Lists all local schemas belonging to the current user and all global schemas.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning XML schema
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)
SCHEMA	XMLTYPE	XML Schema document

DBA_XML_SCHEMAS

Lists all registered local and global schemas in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning XML schema
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)
SCHEMA	XMLTYPE	XML Schema document

DBA_XML_TABLES

Lists all XMLType tables in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

USER_XML_TABLES

Lists all local XMLType tables belonging to the current user.

Column	Datatype	Description
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

ALL_XML_TABLES

Lists all local XMLType tables belonging to the current user and all global tables visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

DBA_XML_TAB_COLS

Lists all XMLType columns in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

USER_XML_TAB_COLS

Lists all XMLType columns in tables belonging to the current user.

Column	Datatype	Description
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

ALL_XML_TAB_COLS

Lists all XMLType columns in tables belonging to the current user and all global tables visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

DBA_XML_VIEWS

Lists all XMLType views in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

USER_XML_VIEWS

Lists all local XMLType views belonging to the current user.

Column	Datatype	Description
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

ALL_XML_VIEWS

Lists all local XMLType views belonging to the current user and all global views visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

DBA_XML_VIEW_COLS

Lists all XMLType columns in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view.
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

USER_XML_VIEW_COLS

Lists all XMLType columns in views belonging to the current user.

Column	Datatype	Description
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.

Column	Datatype	Description
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

ALL_XML_VIEW_COLS

Lists all XMLType columns in views belonging to the current user and all global views visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view.
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

90

DBMS_XPLAN

The DBMS_XPLAN package provides an easy way to format the output of the EXPLAIN PLAN command. For more information on the EXPLAIN PLAN command, see *Oracle9i Database Performance Tuning Guide and Reference.*

This package runs with the privileges of the calling user, not the package owner (SYS).

This chapter discusses the following topics:

Using DBMS_XPLAN

The DBMS_XPLAN package supplies a table function, DISPLAY, to format and display the contents of a plan table, as shown in the following example.

Displaying a Plan Table Using DBMS_XPLAN.DISPLAY: Example

```
Rem
Rem  Execute an explain plan command on a SELECT statement
Rem
EXPLAIN PLAN FOR
SELECT *
FROM emp e, dept d
WHERE e.deptno = d.deptno
  AND e.ename='benoit';

Rem
Rem Display the plan using the DBMS_XPLAN.DISPLAY() table function
Rem
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

This query produces the following output:

```
---------------------------------------------------------------------
| Id  | Operation           |  Name       | Rows  | Bytes | Cost  |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     1 |    50 |     3 |
|*  1 | HASH JOIN           |             |     1 |    50 |     3 |
|*  2 | TABLE ACCESS FULL   | EMP         |     1 |    32 |     1 |
|   3 | TABLE ACCESS FULL   | DEPT        |     4 |    72 |     1 |
---------------------------------------------------------------------
Predicate Information (identified by operation id)
1 - access("E1"."DEPTNO"="D1"."DEPTNO")
2 - filter("E1"."ENAME"='benoit')
```

Summary of DBMS_XPLAN Subprograms

Table 90–1 DBMS_XPLAN Package Subprograms

Subprogram	Description
DISPLAY Function on page 90-3	Displays the contents of the plan table.

DISPLAY Function

This function displays the contents of the plan table.

Syntax

```
DBMS_XPLAN.DISPLAY(
   table_name    IN  VARCHAR2  DEFAULT 'PLAN_TABLE',
   statement_id  IN  VARCHAR2  DEFAULT  NULL,
   format        IN  VARCHAR2  DEFAULT  'TYPICAL');
```

Parameters

Table 90–2 DISPLAY Function Parameters

Parameter	Description
table_name	Specifies the table name where the plan is stored. This parameter defaults to PLAN_TABLE, which is the default plan table for the EXPLAIN PLAN command.
statement_id	Specifies the statement_id of the plan to be displayed. This parameter defaults to NULL, which is the default when the EXPLAIN PLAN command is executed without a set statement_id clause.
format	Controls the level of details for the plan. It accepts four values: ▪ BASIC: Displays the minimum information in the plan—the operation ID, the object name, and the operation option. ▪ TYPICAL: This is the default. Displays the most relevant information in the plan. Partition pruning, parallelism, and predicates are displayed only when available. ▪ ALL: Maximum level. Includes information displayed with the TYPICAL level and adds the SQL statements generated for parallel execution servers (only if parallel). ▪ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.

Displaying Results: Examples

To display the result of the last EXPLAIN PLAN command stored in the plan table:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

To display from other than the default plan table, "my_plan_table":

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY('my_plan_table'));
```

To display the minimum plan information:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY('plan_table', null,
'basic'));
```

To display the plan for a statement identified by 'foo', such as statement_id='foo':

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY('plan_table', 'foo'));
```

Usage Notes

By default, only relevant information is reported by the display table function. In "Displaying a Plan Table Using DBMS_XPLAN.DISPLAY: Example" on page 90-2, the query does not execute in parallel. Hence, information related to the parallelization of the plan is not reported. As shown in the following example, parallel information is reported only if the query executes in parallel.

Displaying a Plan Table with Parallel Information: Example

```
Rem
Rem Execute an explain plan command for a parallel query
Rem
ALTER TABLE emp PARALLEL;
EXPLAIN PLAN for
SELECT * FROM emp e, dept d
   WHERE e.deptno = d.deptno
   AND e.ename    ='benoit'
   ORDER BY e.empno;

Rem
Rem Display the plan using the dbms_xplan.display() table function
Rem
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

The above EXPLAIN PLAN produces output as follows:

```
-------------------------------------------------------------------------------
| Id  | Operation         |Name | Rows | Bytes | Cost | TQ   |IN-OUT|PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |     |    1 |    50 |    3 |67,60 |      |           |
|   1 | SORT ORDER BY    |     |    1 |    50 |    3 |67,61 | P->S | QC(ORDER) |
|   2 | MERGE JOIN       |     |    1 |    50 |    3 |67,62 | P->P | RANGE     |
|   3 | SORT JOIN        |     |    4 |    72 |    3 |67,63 | PCWP |           |
|   4 | TABLE ACCESS FULL|DEPT |    4 |    72 |    2 |67,64 | S->P | BROADCAST |
|*  5 | SORT JOIN        |     |    1 |    32 |    2 |67,65 | PCWP |           |
|*  6 | TABLE ACCESS FULL|EMP  |    1 |    32 |    2 |67,66 | PCWP |           |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id)
   5 - access("E1"."DEPTNO"="D1"."DEPTNO")
       filter("E1"."DEPTNO"="D1"."DEPTNO")
   6 - filter("E1"."ENAME"'benoit')
```

When the query is parallel, information related to parallelism is reported: table queue number (TQ column), table queue type (IN-OUT) and table queue distribution method (PQ Distrib).

By default, if several plans in the plan table match the statement_id parameter passed to the display table function (default value is NULL), only the plan corresponding to the last EXPLAIN PLAN command is displayed. Hence, there is no need to purge the plan table after each EXPLAIN PLAN. However, you should purge the plan table regularly (for example, by using the TRUNCATE TABLE command) to ensure good performance in the execution of the DISPLAY table function.

For ease of use, you can define a view on top of the display table function and then use that view to display the output of the EXPLAIN PLAN command, as shown below:

Using a View to Display Output: Example

```
# define plan view
create view plan as select * from table(dbms_xplan.display);

# display the output of the last explain plan command
select * from plan;
```

91

DBMS_XSLPROCESSOR

With DBMS_XSLPROCESSOR, you can access the contents and structure of XML documents.

> **See Also:** *Oracle9i XML API Reference - XDK and Oracle XML DB* for more information

This chapter details the following:

- Subprograms of DBMS_XSLPROCESSOR

Description of DBMS_XSLPROCESSOR

The Extensible Stylesheet Language Transformation (XSLT), describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet. The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree. This PL/SQL implementation of the XSL processor followed the W3C XSLT working draft (rev WD-xslt-19990813) and included the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformation it must effect.

The following is the default behavior for this PL/SQL XSL Processor:

- A result tree which can be accessed by DOM APIs is built
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Subprograms of DBMS_XSLPROCESSOR

Table 91-1: Summary of Subprograms of DBMS_XSLPROCESSOR

Subprogram	Description
newProcessor() on page 91-3	Returns a new processor instance.
processXSL() on page 91-3	Transforms an input XML document.
showWarnings() on page 91-5	Turns warnings on or off.
setErrorLog() on page 91-6	Sets errors to be sent to the specified file.
newStylesheet() on page 91-6	Creates a new stylesheet using the given input and reference URLs.
transformNode() on page 91-7	Transforms a node in a DOM tree using the given stylesheet.
selectNodes() on page 91-7	Selects nodes from a DOM tree that match the given pattern.
selectSingleNodes() on page 91-8	Selects the first node from the tree that matches the given pattern.
valueOf() on page 91-8	Retrieves the value of the first node from the tree that matches the given pattern
setParam() on page 91-8	Sets a top-level parameter in the stylesheet
removeParam() on page 91-9	Removes a top-level stylesheet parameter

Table 91-1: Summary of Subprograms of DBMS_XSLPROCESSOR (Cont.)

Subprogram	Description
resetParams() on page 91-9	Resets the top-level stylesheet parameters
freeStylesheet() on page 91-9	Frees a stylesheet object
freeProcessor() on page 91-10	Frees a processor object

newProcessor()

Returns a new processor instance. This function must be called before the default behavior of Processor can be changed and if other processor methods need to be used.

Syntax

```
FUNCTION newProcessor RETURN Processor;
```

processXSL()

Transforms input XML document. Any changes to the default processor behavior should be effected before calling this procedure. An application error is raised if processing fails. The options are described in the following table.

Syntax	Description
FUNCTION processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument), RETURN DOMDocumentFragment;	Transforms input XML document using given DOMDocument and stylesheet, and returns the resultant document fragment.
FUNCTION processXSL(p Processor, ss Stylesheet, url VARCHAR2, RETURN DOMDocumentFragment;	Transforms input XML document using given document as URL and the stylesheet, and returns the resultant document fragment.

Syntax	Description
FUNCTION processXSL(p Processor, ss Stylesheet, clb CLOB) RETURN DOMDocumentFragment;	Transforms input XML document using given document as CLOB and the stylesheet, and returns the resultant document fragment.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML document using given DOMDocument and the stylesheet, and writes the output to the specified file.
PROCEDURE processXSL(p Processor, ss Stylesheet, url VARCHAR2, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML document using given URL and the stylesheet, and writes the output to the specified file in a specified directory.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument, cl IN OUT CLOB);	Transforms input XML document using given DOMDocument and the stylesheet, and writes the output to a CLOB.
FUNCTION processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment) RETURN DOMDocumentFragment;	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and returns the resultant document fragment.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to the specified file in a specified directory.

Syntax	Description
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, buf IN OUT VARCHAR2);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to a buffer.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, cl IN OUT CLOB);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to a CLOB.

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
ss	(IN)	Stylesheet instance.
xmldoc	(IN)	XML document being transformed.
url	(IN)	URL for the information being transformed.
clb	(IN)	CLOB containing information to be transformed.
dir	(IN)	Directory where processing output file is saved.
fileName	(IN)	Processing output file.
cl	(IN/OUT)	CLOB to which the processing output is saved.
buf	(IN/OUT)	Buffer to which the processing output is saved.
xmldf	(IN)	XML document fragment being transformed.

showWarnings()

Turns warnings on (TRUE) or off (FALSE).

Syntax

```
PROCEDURE showWarnings( p Processor,
                        yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
yes	(IN)	Mode to set: TRUE to show warnings, FALSE otherwise

setErrorLog()

Sets errors to be sent to the specified file.

Syntax

```
PROCEDURE setErrorLog( p Processor,
                       fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
fileName	(IN)	complete path of the file to use as the error log.

newStylesheet()

Creates and returns a new stylesheet instance. The options are described in the following table.

Syntax	Description
FUNCTION newStylesheet(xmldoc DOMDocument, ref VARCHAR2) RETURN Stylesheet;	Creates and returns a new stylesheet instance using the given DOMDocument and reference URLs.
FUNCTION newStylesheet(inp VARCHAR2, ref VARCHAR2) RETURN Stylesheet;	Creates and returns a new stylesheet instance using the given input and reference URLs.

Parameter	IN / OUT	Description
xmldoc	(IN)	DOMDocument to use for construction.

Parameter	IN / OUT	Description
inp	(IN)	Input URL to use for construction.
ref	(IN)	Reference URL

transformNode()

Transforms a node in a DOM tree using the given stylesheet, and returns the result of the transformation as a DOMDocumentFragment.

Syntax

```
FUNCTION transformNode( n DOMNode,
                        ss Stylesheet)
                        RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to transform.
ss	(IN)	Stylesheet to use.

selectNodes()

Selects nodes which match the given pattern from a DOM tree, and returns the result of the selection.

Syntax

```
FUNCTION selectNodes( n DOMNode,
                      pattern VARCHAR2)
                      RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.

selectSingleNodes()

Selects the first node from the tree that matches the given pattern, and returns that node.

Syntax

```
FUNCTION selectSingleNodes( n DOMNode,
                            pattern VARCHAR2)
                            RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.

valueOf()

Retrieves the value of the first node from the tree that matches the given pattern.

Syntax

```
PROCEDURE valueOf( n DOMNode,
                   pattern VARCHAR2,
                   val OUT VARCHAR2);
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.
val	(OUT)	Retrieved value.

setParam()

Sets a top level parameter in the stylesheet. The parameter value must be a valid XPath expression. Literal string values must be quoted.

Syntax

```
PROCEDURE setParam( ss Stylesheet,
                    name VARCHAR2,
```

```
                        value VARCHAR2);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.
name	(IN)	Name of the parameter.
value	(IN)	Value of the parameter.

removeParam()

Removes a top level stylesheet parameter.

Syntax

```
PROCEDURE removeParam( ss Stylesheet,
                       name VARCHAR2);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.
name	(IN)	Name of the parameter.

resetParams()

Resets the top-level stylesheet parameters.

Syntax

```
PROCEDURE resetParams( ss Stylesheet);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.

freeStylesheet()

Frees a Stylesheet object.

Syntax

```
PROCEDURE freestylesheet( ss Stylesheet);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.

freeProcessor()

Frees a Processor object.

Syntax

```
PROCEDURE freeProccessor( p Processor);
```

Parameter	IN / OUT	Description
p	(IN)	Processor.

92

DEBUG_EXTPROC

The DEBUG_EXTPROC package enables you to start up the extproc agent within a session. This utility package can help you debug external procedures.

This chapter discusses the following topics:

- Requirements and Installation Notes for DEBUG_EXTPROC
- Using DEBUG_EXTPROC
- Summary of DBMS_EXTPROC Subprograms

Requirements and Installation Notes for DEBUG_EXTPROC

Requirements

Your Oracle account must have EXECUTE privileges on the package and CREATE LIBRARY privileges.

Note: DEBUG_EXTPROC works only on platforms with debuggers that can attach to a running process.

Installation Notes

To install the package, run the script DBGEXTP.SQL.

- Install/load this package in the Oracle USER where you want to debug the 'extproc' process.
- Ensure that you have execute privileges on package DEBUG_EXTPROC

```
SELECT SUBSTR(OBJECT_NAME, 1, 20)
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'DEBUG_EXTPROC';
```

- You can install this package as any other user, as long as you have EXECUTE privileges on the package.

Using DEBUG_EXTPROC

Usage Assumptions

This assumes that the Listener has been appropriately configured to startup an external procedures 'extproc' agent.

This also assumes that you built your shared library with debug symbols to aid in the debugging process. Please check the C compiler manual pages for the appropriate C compiler switches to build the shared library with debug symbols.

Usage Notes

- Start a brand new oracle session through SQL*Plus or OCI program by connecting to ORACLE.
- Execute procedure DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT to startup the extproc agent in this session; for example, execute DEBUG_EXTPROC.STARTUP_

EXTPROC_AGENT; Do not exit this session, because that terminates the extproc agent.

- Determine the PID of the extproc agent that was started up for this session.
- Using a debugger (for example, gdb, dbx, or the native system debugger), load the extproc executable and attach to the running process.
- Set a breakpoint on function 'pextproc' and let the debugger continue with its execution.
- Now execute your external procedure in the same session where you first executed DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT
- Your debugger should now break in function 'pextproc'. At this point in time, the shared library referenced by your PL/SQL external function would have been loaded and the function resolved. Now set a breakpoint in your C function and let the debugger continue its execution.

Because PL/SQL loads the shared library at runtime, the debugger you use may or may not automatically be able to track the new symbols from the shared library. You may have to issue some debugger command to load the symbols (for example, 'share' in gdb)

- The debugger should now break in your C function. Its assumed that you had built the shared library with debugging symbols.
- Now proceed with your debugging.

Summary of DBMS_EXTPROC Subprograms

DEBUG_EXTPROC contains one subprogram: STARTUP_EXTPROC_AGENT procedure. This starts up the extproc agent process in the session

STARTUP_EXTPROC_AGENT Procedure

This procedure starts up the extproc agent process in the session.This enables you to get the PID of the executing process. This PID is needed to be able to attach to the running process using a debugger.

Syntax

```
DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT;
```

93

UTL_COLL

The UTL_COLL package lets PL/SQL programs use collection locators to query and update.

This chapter discusses the following topics:

- Summary of UTL_COLL Subprograms

Summary of UTL_COLL Subprograms

There is currently only one function supported in this package:IS_LOCATOR.

IS_LOCATOR Function

This function determines whether a collection item is actually a locator or not.

Syntax

```
UTL_COLL.IS_LOCATOR (
   collection IN ANY)
  RETURNS BOOLEAN;
```

Parameters

Table 93–1 IS_LOCATOR Function Parameters

Parameter	Description
collection	Nested table or varray item.

Returns

Table 93–2 IS_LOCATOR Function Returns

Return Value	Description
1	Collection item is indeed a locator.
0	Collection item is not a locator.

Pragmas

Asserts WNDS, WNPS and RNPS pragmas

Example

```
CREATE OR REPLACE TYPE list_t as TABLE OF VARCHAR2(20);
/

CREATE OR REPLACE TYPE phone_book_t AS OBJECT (
  pno  number,
  ph   list_t );
/
```

```
CREATE TABLE phone_book OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph;
CREATE TABLE phone_book1 OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph_1 RETURN LOCATOR;

INSERT INTO phone_book VALUES(1, list_t('650-633-5707','650-323-0953'));
INSERT INTO phone_book1 VALUES(1, list_t('415-555-1212'));

CREATE OR REPLACE PROCEDURE chk_coll IS
  plist list_t;
  plist1 list_t;
BEGIN
  SELECT ph INTO plist FROM phone_book WHERE pno=1;

  SELECT ph INTO plist1 FROM phone_book1 WHERE pno=1;

  IF (UTL_COLL.IS_LOCATOR(plist)) THEN
    DBMS_OUTPUT.PUT_LINE('plist is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist is not a locator');
  END IF;

  IF (UTL_COLL.IS_LOCATOR(plist1)) THEN
    DBMS_OUTPUT.PUT_LINE('plist1 is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist1 is not a locator');
  END IF;

END chk_coll;

SET SERVEROUTPUT ON
EXECUTE chk_coll;
```

94

UTL_ENCODE

The UTL_ENCODE package provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts. You can use UTL_ENCODE functions to encode the body of email text. The package also contains the decode counterpart functions of the encode functions. The functions follow published standards for encoding to accommodate non-Oracle utilities on the sending or receiving ends.

This chapter discusses the following topics:

Summary of UTL_ENCODE Subprograms

Table 94–1 UTL_ENCODE Subprograms

Subprogram	Description
BASE64_ENCODE Function on page 94-2	Encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string
BASE64_DECODE Function on page 94-3	Reads the base 64-encoded RAW input string and decodes it to its original RAW value
UUENCODE Function on page 94-4	Reads the RAW input string and encodes it to the corresponding uuencode format string
UUDECODE Function on page 94-5	Reads the RAW uuencode format input string and decodes it to the corresponding RAW string
QUOTED_PRINTABLE_ ENCODE Function on page 94-6	Reads the RAW input string and encodes it to the corresponding quoted printable format string
QUOTED_PRINTABLE_ DECODE Function on page 94-6	Reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string

BASE64_ENCODE Function

This function encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string.

Syntax

```
UTL_ENCODE.BASE64_ENCODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(base64_encode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 94–2 BASE64_ENCODE Function Parameters

Parameter	Description
r	The RAW value to be encoded. There are no defaults or optional parameters.

Returns

Table 94–3 BASE64_ENCODE Function Returns

Return	Description
RAW	Contains the encoded base 64 elements

BASE64_DECODE Function

This function reads the base 64-encoded RAW input string and decodes it to its original RAW value.

Syntax

```
UTL_ENCODE.BASE64_DECODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(base64_decode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 94–4 BASE64_DECODE Function Parameters

Parameter	Description
r	The RAW string containing base 64-encoded data. There are no defaults or optional parameters.

Returns

Table 94–5 BASE64_DECODE Function Returns

Return	Description
RAW	Contains the decoded string

UUENCODE Function

This function reads the RAW input string and encodes it to the corresponding uuencode format string. The output of this function is cumulative, in that it can be used to encode large data streams, by splitting the data stream into acceptably sized RAW values, encoded, and concatenated into a single encoded string. Also see "UUDECODE Function" on page 94-5.

Syntax

```
UTL_ENCODE.UUENCODE (
   r          IN RAW,
   type       IN PLS_INTEGER DEFAULT 1,
   filename   IN VARCHAR2 DEFAULT NULL,
   permission IN VARCHAR2 DEFAULT NULL) RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(uuencode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 94–6 UUENCODE Function Parameters

Parameter	Description
r	RAW string
type	Optional number parameter containing the type of uuencoded output. Options: complete—a defined PL/SQL constant with a value of 1. (default) header_piece middle_piece end_piece
filename	Optional varchar2 parameter containing the uuencode filename; the default is uuencode.txt

Table 94–6 UUENCODE Function Parameters

Parameter	Description
permission	Optional varchar2 parameter containing the permission mode; the default is 0 (a text string zero).

Returns

Table 94–7 UUENCODE Function Returns

Return	Description
RAW	Contains the uuencode format string

UUDECODE Function

This function reads the RAW uuencode format input string and decodes it to the corresponding RAW string. See "UUENCODE Function" on page 94-4 for discussion of the cumulative nature of UUENCODE and UUDECODE for data streams.

Syntax

```
UTL_ENCODE.UUDECODE (
   r  IN RAW)
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(uudecode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 94–8 DUDECODE Function Parameters

Parameter	Description
r	The RAW string containing the uuencoded data string. There are no defaults or optional parameters.

Returns

Table 94–9 UUDECODE Function Returns

Return	Description
RAW	The decoded RAW string

QUOTED_PRINTABLE_ENCODE Function

This function reads the RAW input string and encodes it to the corresponding quoted printable format string.

Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_ENCODE (
   r  IN RAW
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_encode, WNDS, RNDS,WNPS, RNPS);
```

Parameters

Table 94–10 QUOTED_PRINTABLE_ENCODE Function Parameters

Parameter	Description
r	The RAW string. There are no defaults or optional parameters.

Returns

Table 94–11 QUOTED_PRINTABLE_ENCODE Function Returns

Return	Description
RAW	Contains the quoted printable string

QUOTED_PRINTABLE_DECODE Function

This function reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string.

Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_DECODE (
   r  IN RAW
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_decode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 94–12 QUOTED_PRINTABLE_DECODE Function Parameters

Parameters	Description
r	The RAW string containing a quoted printable data string. There are no defaults or optional parameters.

Returns

Table 94–13 QUOTED_PRINTABLE_DECODE Function Returns

Return	Description
RAW	The decoded string

95

UTL_FILE

With the UTL_FILE package, your PL/SQL programs can read and write operating system text files. UTL_FILE provides a restricted version of operating system stream file I/O.

UTL_FILE I/O capabilities are similar to standard operating system stream file I/O (OPEN, GET, PUT, CLOSE) capabilities, but with some limitations. For example, you call the FOPEN function to return a file handle, which you use in subsequent calls to GET_LINE or PUT to perform stream I/O to a file. When file I/O is done, you call FCLOSE to complete any output and free resources associated with the file.

> **Note:** The UTL_FILE package is similar to the client-side TEXT_IO package currently provided by Oracle Procedure Builder. Restrictions for a server implementation require some API differences between UTL_FILE and TEXT_IO. In PL/SQL file I/O, errors are returned using PL/SQL exceptions.

This chapter discusses the following topics:

Security

UTL_FILE is available for both client-side and server-side PL/SQL. The client implementation (text I/O) is subject to normal operating system file permission checking. However, the server implementation may be running in a privileged mode, which requires a restriction on the directories that you can access.

In the past, accessible directories for the UTL_FILE functions were specified in the initialization file using the UTL_FILE_DIR parameter. However, UTL_FILE_DIR access is not recommended. It is recommended that you use the CREATE DIRECTORY feature, which replaces UTL_FILE_DIR. Directory objects offer more flexibility and granular control to the UTL_FILE application administrator, can be maintained dynamically (that is, without shutting down the database), and are consistent with other Oracle tools. CREATE DIRECTORY privilege is granted only to SYS and SYSTEM by default.

Note: use the CREATE DIRECTORY feature instead of UTL_FILE_DIR for directory access verification.

File Ownership and Protections

On UNIX systems, the owner of a file created by the FOPEN function is the owner of the shadow process running the instance. Normally, this owner is ORACLE. Files created using FOPEN are always writable and readable using the UTL_FILE subprograms, but nonprivileged users who need to read these files outside of PL/SQL may need access from a system administrator.

Examples (UNIX-Specific)

Given the following:

```
SQL> CREATE DIRECTORY log_dir AS '/appl/gl/log';
SQL> GRANT READ ON DIRECTORY log_dir TO DBA;

SQL> CREATE DIRECTORY out_dir AS '/appl/gl/user'';
SQL> GRANT READ ON DIRECTORY user_dir TO PUBLIC;
```

The following file locations and filenames are valid and accessible as follows:

File Location	Filename	Accessible By
/appl/gl/log	L12345.log	Users with DBA privilege

File Location	Filename	Accessible By
/appl/gl/user	u12345.tmp	All users

The following file locations and filenames are invalid:

File Location	Filename	Invalid Because
/appl/gl/log/backup	L12345.log	# subdirectories are not accessible
/APPL/gl/log	L12345.log	# directory strings must follow case sensitivity rules as required by the O/S
/appl/gl/log	backup/L1234.log	# filenames may not include portions of directory paths
/user/tmp	L12345.log	# no corresponding CREATE DIRECTORY command has been issued

Caution: There are no user-level file permissions. UTL_FILE directory object privileges give you read and write access to all files within the specified directory.

Exceptions

Table 95–1 UTL_FILE Package Exceptions

Exception Name	Description
INVALID_PATH	File location is invalid.
INVALID_MODE	The open_mode parameter in FOPEN is invalid.
INVALID_FILEHANDLE	File handle is invalid.
INVALID_OPERATION	File could not be opened or operated on as requested.
READ_ERROR	Operating system error occurred during the read operation.
WRITE_ERROR	Operating system error occurred during the write operation.
INTERNAL_ERROR	Unspecified PL/SQL error

Table 95–1 UTL_FILE Package Exceptions

Exception Name	Description
CHARSETMISMATCH	A file is opened using FOPEN_NCHAR, but later I/O operations use nonchar functions such as PUTF or GET_LINE.
FILE_OPEN	The requested operation failed because the file is open.
INVALID_MAXLINESIZE	The MAX_LINESIZE value for FOPEN() is invalid; it should be within the range 1 to 32767.
INVALID_FILENAME	The filename parameter is invalid.
ACCESS_DENIED	Permission to access to the file location is denied.
INVALID_OFFSET	The ABSOLUTE_OFFSET parameter for FSEEK() is invalid; it should be greater than 0 and less than the total number of bytes in the file.
DELETE_FAILED	The requested file delete operation failed.
RENAME_FAILED	The requested file rename operation failed.

Procedures in UTL_FILE can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

Types

The contents of FILE_TYPE are private to the UTL_FILE package. You should not reference or change components of this record.

```
TYPE file_type IS RECORD (
   id       BINARY_INTEGER,
   datatype BINARY_INTEGER);
```

Summary of UTL_FILE Subprograms

Table 95–2 UTL_FILE Subprograms

Subprogram	Description
FOPEN Function on page 95-6	Opens a file for input or output.
FOPEN_NCHAR Function on page 95-7	Opens a file in Unicode for input or output.

Table 95–2 UTL_FILE Subprograms

Subprogram	Description
IS_OPEN Function on page 95-8	Determines if a file handle refers to an open file.
FCLOSE Procedure on page 95-9	Closes a file.
FCLOSE_ALL Procedure on page 95-9	Closes all open file handles.
GET_LINE Procedure on page 95-10	Reads text from an open file.
GET_LINE_NCHAR Procedure on page 95-11	Reads text in Unicode from an open file.
GET_RAW Function on page 95-12	Reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read.
PUT Procedure on page 95-12	Writes a string to a file.
PUT_NCHAR Procedure on page 95-13	Writes a Unicode string to a file.
PUT_RAW Function on page 95-14	Accepts as input a RAW data value and writes the value to the output buffer.
NEW_LINE Procedure on page 95-15	Writes one or more operating system-specific line terminators to a file.
PUT_LINE Procedure on page 95-15	Writes a line to a file. This appends an operating system-specific line terminator.
PUT_LINE_NCHAR Procedure on page 95-16	Writes a Unicode line to a file.
PUTF Procedure on page 95-17	A PUT procedure with formatting.
PUTF_NCHAR Procedure on page 95-18	A PUT_NCHAR procedure with formatting. Writes a Unicode string to a file, with formatting.
FFLUSH Procedure on page 95-19	Physically writes all pending output to a file.
FSEEK Function on page 95-20	Adjusts the file pointer forward or backward within the file by the number of bytes specified.
FREMOVE Function on page 95-21	Deletes a disk file, assuming that you have sufficient privileges.

Table 95–2 UTL_FILE Subprograms

Subprogram	Description
FCOPY Function on page 95-21	Copies a contiguous portion of a file to a newly created file.
FGETPOS Function on page 95-22	Returns the current relative offset position within a file, in bytes.
FGETATTR Procedure on page 95-23	Reads and returns the attributes of a disk file.
FRENAME Function on page 95-24	Renames an existing file to a new name, similar to the Unix `mv` function.

Note: The file location and file name parameters are supplied to the `FOPEN` function as separate strings, so that the file location can be checked against the list of accessible directories as specified by the `ALL_DIRECTORIES` view of accessible directory objects. Together, the file location and name must represent a legal filename on the system, and the directory must be accessible. A subdirectory of an accessible directory is not necessarily also accessible; it too must be specified using a complete path name matching an `ALL_DIRECTORIES` object.

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

FOPEN Function

This function opens a file. You can specify the maximum line size and have a maximum of 50 files open simultaneously. See also "FOPEN_NCHAR Function" on page 95-7.

Syntax

```
UTL_FILE.FOPEN (
   location     IN VARCHAR2,
   filename     IN VARCHAR2,
   open_mode    IN VARCHAR2,
   max_linesize IN BINARY_INTEGER)
  RETURN file_type;
```

Parameters

Table 95–3 FOPEN Function Parameters

Parameter	Description
location	Directory location of file.
filename	File name, including extension (file type), without directory path. In Unix, the filename cannot end with /.
open_mode	Specifies how the file is opened. Modes include: r–read text w–write text a–append text If you try to open a file that does not exist using a value for open_mode, then the file is created in write mode.
max_linesize	Maximum number of characters per line, including the newline character, for this file. (minimum value 1, maximum value 32767). The default is approximately 1000 bytes.

Returns

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL_FILE package, and individual components should not be referenced or changed by the UTL_FILE user.

Table 95–4 FOPEN Function Returns

Return	Description
file_type	Handle to open file.

Exceptions

```
INVALID_PATH: File location or name was invalid.
INVALID_MODE: The open_mode string was invalid.
INVALID_OPERATION: File could not be opened as requested.
INVALID_MAXLINESIZE: Specified max_linesize is too large or too small.
```

FOPEN_NCHAR Function

This function opens a file in Unicode for input or output, with the maximum line size specified. You can have a maximum of 50 files open simultaneously. With this

function, you can read or write a text file in Unicode instead of in the database charset. See also FOPEN Function on page 95-6.

Syntax

```
UTL_FILE.FOPEN_NCHAR (
   location     IN VARCHAR2,
   filename     IN VARCHAR2,
   open_mode    IN VARCHAR2,
   max_linesize IN BINARY_INTEGER)
RETURN file_type;
```

Parameters

Table 95–5 FOPEN_NCHAR Function Parameters

Parameter	Description
`location`	Directory location of file.
`filename`	File name (including extension).
`open_mode`	Open mode (`r`, `w`, `a`).
`max_linesize`	Maximum number of characters per line, including the newline character, for this file. (minimum value 1, maximum value 32767).

IS_OPEN Function

This function tests a file handle to see if it identifies an open file. `IS_OPEN` reports only whether a file handle represents a file that has been opened, but not yet closed. It does not guarantee that there will be no operating system errors when you attempt to use the file handle.

Syntax

```
UTL_FILE.IS_OPEN (
   file  IN FILE_TYPE)
  RETURN BOOLEAN;
```

Parameters

Table 95–6 IS_OPEN Function Parameters

Parameter	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.

Returns

TRUE or FALSE

Exceptions

None.

FCLOSE Procedure

This procedure closes an open file identified by a file handle. If there is buffered data yet to be written when FCLOSE runs, then you may receive a WRITE_ERROR exception when closing a file.

Syntax

```
UTL_FILE.FCLOSE (
   file IN OUT FILE_TYPE);
```

Parameters

Table 95–7 FCLOSE Procedure Parameters

Parameter	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.

Exceptions

```
WRITE_ERROR
INVALID_FILEHANDLE
```

FCLOSE_ALL Procedure

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception.

Note: FCLOSE_ALL does not alter the state of the open file handles held by the user. This means that an IS_OPEN test on a file handle after an FCLOSE_ALL call still returns TRUE, even though the file has been closed. No further read or write operations can be performed on a file that was open before an FCLOSE_ALL.

Syntax

```
UTL_FILE.FCLOSE_ALL;
```

Parameters

None.

Exceptions

WRITE_ERROR

GET_LINE Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to, but not including, the line terminator, or up to the end of the file, or up to the end of the linesize parameter. It cannot exceed the max_linesize specified in FOPEN.

If the line does not fit in the buffer, then a VALUE_ERROR exception is raised. If no text was read due to end of file, then the NO_DATA_FOUND exception is raised.

Because the line terminator character is not read into the buffer, reading blank lines returns empty strings.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. The default is approximately 1000 bytes, depending on your platform. See also "GET_LINE_NCHAR Procedure" on page 95-11.

Syntax

```
UTL_FILE.GET_LINE (
   file        IN  FILE_TYPE,
   buffer      OUT VARCHAR2,
   linesize    IN  NUMBER,
   len         IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 95–8 GET_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call. The file must be open for reading (mode r), otherwise an INVALID_OPERATION exception is raised.
buffer	Data buffer to receive the line read from the file.
linesize	Specifies the maximum number of bytes to read.
len	The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
READ_ERROR
NO_DATA_FOUND
VALUE_ERROR

GET_LINE_NCHAR Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. With this function, you can read a text file in Unicode instead of in the database charset. See also "GET_LINE Procedure" on page 95-10.

Syntax

```
UTL_FILE.GET_LINE_NCHAR (
   file        IN  FILE_TYPE,
   buffer      OUT NVARCHAR2,
   len         IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 95–9 GET_LINE_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
buffer	Data buffer to receive the line read from the file.
len	The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.

GET_RAW Function

This function reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read.

Syntax

```
UTL_FILE.GET_RAW (
   fid  IN  utl_file.file_type,
   r    OUT NOCOPY RAW,
   len  IN  PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 95–10 GET_RAW Procedure Parameters

Parameters	Description
fid	The file ID.
r	The RAW data.
len	The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.

PUT Procedure

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. No line terminator is appended by PUT; use NEW_LINE to terminate the line or use PUT_LINE to write a

complete line with a line terminator. See also "PUT_NCHAR Procedure" on page 95-13.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.

Syntax

```
UTL_FILE.PUT (
   file      IN FILE_TYPE,
   buffer    IN VARCHAR2);
```

Parameters

Table 95–11 PUT Procedure Parameters

Parameters	Description
`file`	Active file handle returned by an `FOPEN_NCHAR` call. The file must be open for reading (mode `r`). If the file is opened by `FOPEN` instead of `FOPEN_NCHAR`, a `CHARSETMISMATCH` exception is raised.
`buffer`	Buffer that contains the text to be written to the file. You must have opened the file using mode `w` or mode `a`; otherwise, an `INVALID_OPERATION` exception is raised.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database charset. See also "PUT Procedure" on page 95-12.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.

Syntax

```
UTL_FILE.PUT_INCHAR (
   file      IN FILE_TYPE,
   buffer    IN NVARCHAR2);
```

Parameters

Table 95–12 PUT_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
buffer	Buffer that contains the text to be written to the file. You must have opened the file using mode w or mode a; otherwise, an INVALID_OPERATION exception is raised.

PUT_RAW Function

This function accepts as input a RAW data value and writes the value to the output buffer. You can request an automatic flush of the buffer by setting the third argument to TRUE.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Syntax

```
UTL_FILE. PUT_RAW (
   fid       IN utl_file.file_type,
   r         IN RAW,
   autoflush IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 95–13 PUT_RAW Procedure Parameters

Parameters	Description
fid (IN)	The file ID.

Table 95–13 PUT_RAW Procedure Parameters

Parameters	Description
r (IN)	The RAW data written to the buffer.
autoflush (IN)	If TRUE, performs a flush after writing the value to the output buffer; default is FALSE.

NEW_LINE Procedure

This procedure writes one or more line terminators to the file identified by the input file handle. This procedure is separate from PUT because the line terminator is a platform-specific character or sequence of characters.

Syntax

```
UTL_FILE.NEW_LINE (
   file     IN FILE_TYPE,
   lines    IN NATURAL := 1);
```

Parameters

Table 95–14 NEW_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.
lines	Number of line terminators to be written to the file.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT_LINE Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. PUT_LINE terminates the line with the platform-specific line terminator character or characters.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

See also "PUT_LINE_NCHAR Procedure" on page 95-16.

Syntax

```
UTL_FILE.PUT_LINE (
   file      IN FILE_TYPE,
   buffer    IN VARCHAR2,
   autoflush IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 95–15 PUT_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
buffer	Text buffer that contains the lines to be written to the file.
autoflush	Flushes the buffer to disk after the WRITE.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUT_LINE_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database charset. See also "PUT_LINE Procedure" on page 95-15.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Syntax

```
UTL_FILE.PUT_LINE_NCHAR (
```

```
   file    IN FILE_TYPE,
   buffer  IN NVARCHAR2);
```

Parameters

Table 95–16 PUT_LINE_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
buffer	Text buffer that contains the lines to be written to the file.

PUTF Procedure

This procedure is a formatted PUT procedure. It works like a limited printf(). The format string can contain any text, but the character sequences %s and \n have special meaning.

Character Sequence	Meaning
%s	Substitute this sequence with the string value of the next argument in the argument list.
\n	Substitute with the appropriate platform-specific line terminator.

See also "PUTF_NCHAR Procedure" on page 95-18.

Syntax

```
UTL_FILE.PUTF (
   file    IN FILE_TYPE,
   format  IN VARCHAR2,
   [arg1   IN VARCHAR2  DEFAULT NULL,
   . . .
   arg5    IN VARCHAR2  DEFAULT NULL]);
```

Parameters

Table 95–17 PUTF Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
format	Format string that can contain text as well as the formatting characters \n and %s.
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.

Example

The following example writes the lines:

```
Hello, world!
I come from Zork with greetings for all earthlings.

my_world  varchar2(4) := 'Zork';
...
PUTF(my_handle, 'Hello, world!\nI come from %s with %s.\n',
                my_world,
                'greetings for all earthlings');
```

If there are more %s formatters in the format parameter than there are arguments, then an empty string is substituted for each %s for which there is no matching argument.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUTF_NCHAR Procedure

This procedure is a formatted PUT_NCHAR procedure. With this function, you can write a text file in Unicode instead of in the database charset. See also "PUTF Procedure" on page 95-17. See also "PUT_LINE Procedure" on page 95-15.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. The default is approximately 1000 bytes, depending on your platform. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Syntax

```
UTL_FILE.PUTF_NCHAR (
   file    IN FILE_TYPE,
   format  IN NVARCHAR2,
   [arg1   IN NVARCHAR2  DEFAULT NULL,
   . . .
   arg5    IN NVARCHAR2  DEFAULT NULL]);
```

Parameters

Table 95–18 PUTF_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
format	Format string that can contain text as well as the formatting characters \n and %s.
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.

FFLUSH Procedure

FFLUSH physically writes pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The FFLUSH procedure forces the buffered data to be written to the file. The data must be terminated with a newline character.

Flushing is useful when the file must be read while still open. For example, debugging messages can be flushed to the file so that they can be read immediately.

Syntax

```
UTL_FILE.FFLUSH (
   file  IN FILE_TYPE);
invalid_maxlinesize  EXCEPTION;
```

Parameters

Table 95–19 FFLUSH Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

FSEEK Function

This function adjusts the file pointer forward or backward within the file by the number of bytes specified.

If offset, the function seeks to a byte offset. If the end of the file or the beginning of the file is reached before seeking is done, the function returns the last or first row, respectively.

If loc, the function seeks to an absolute location specified in bytes.

Syntax

```
UTL_FILE.FSEEK (
   fid             IN utl_file.file_type,
   absolute_offset IN PL_INTEGER DEFAULT NULL,
   relative_offset IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 95–20 FSEEK Procedure Parameters

Parameters	Description
fid (in)	The file ID.

Table 95–20 FSEEK Procedure Parameters

Parameters	Description
absolute_offset (IN)	The absolute location to which to seek; default = NULL
relative_offset (IN)	The number of bytes to seek forward or backward; positive = forward, negative integer = backward, zero = current position, default = NULL

Notes

Using this function, you can read previous lines in the file without first closing and reopening the file. You must know the number of bytes by which you want to navigate.

FREMOVE Function

This function deletes a disk file, assuming that you have sufficient privileges.

Syntax

```
UTL_FILE.FREMOVE (
   location IN VARCHAR2,
   filename IN VARCHAR2);
```

Parameters

Table 95–21 FREMOVE Procedure Parameters

Parameters	Description
location (IN)	The directory location of the file, a DIRECTORY_NAME from ALL_DIRECTORIES (case sensitive)
filename (IN)	The name of the file to be deleted

Notes

The FREMOVE function does not verify privileges prior to deleting the file. The O/S verifies file and directory permissions. An exception is returned on failure.

FCOPY Function

This function copies a contiguous portion of a file to a newly created file. By default, the whole file is copied if the start_line and end_line parameters are omitted.

The source file is opened in read mode. The destination file is opened in write mode. A starting and ending line number can optionally be specified to select a portion from the center of the source file for copying.

Syntax

```
UTL_FILE.FCOPY (
   location   IN VARCHAR2,
   filename   IN VARCHAR2,
   dest_dir   IN VARCHAR2,
   dest_file  IN VARCHAR2,
   start_line IN PLS_INTEGER DEFAULT 1,
   end_line   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 95–22 FCOPY Procedure Parameters

Parameters	Description
`location (IN)`	The directory location of the source file, a `DIRECTORY_NAME` from the `ALL_DIRECTORIES` view (case sensitive)
`filename (IN)`	The source file to be copied
`dest_dir (IN)`	The destination directory where the destination file is created.
`dest_file (N)`	The destination file created from the source file.
`start_line (IN)`	The line number at which to begin copying. The default is `1` for the first line.
`end_line (IN)`	The line number at which to stop copying. The default is `NULL`, signifying end of file.

FGETPOS Function

This function returns the current relative offset position within a file, in bytes.

Syntax

```
UTL_FILE.FGETPOS (
   fileid IN file_type)
 RETURN PLS_INTEGER;
```

Parameters

Table 95–23 FGETPOS Parameters

Parameters	Description
fileid (IN)	The directory location of the source file

Returns

FGETPOS returns the relative offset position for an open file, in bytes. It raises an exception if the file is not open. It returns 0 for the beginning of the file.

FGETATTR Procedure

This procedure reads and returns the attributes of a disk file.

Syntax

```
UTL_FILE.FGETATTR(
   location    IN VARCHAR2,
   filename    IN VARCHAR2,
   exists      OUT BOOLEAN,
   file_length OUT NUMBER,
   blocksize   OUT NUMBER);
```

Parameters

Table 95–24 FGETATTR Procedure Parameters

Parameters	Description
location	Directory location of the source file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)
filename	The name of the source file to be copied
exists	A BOOLEAN for whether or not the file exists
file_length	The length of the file in bytes. NULL if file does not exist.
blocksize	The file system block size in bytes. NULL if the file does not exist.

FRENAME Function

This function renames an existing file to a new name, similar to the Unix mv function. Permission on both the source and destination directories must be granted. You can use the overwrite parameter to specify whether or not to overwrite a file if one exists in the destination directory. The default is FALSE for no overwrite.

Syntax

```
UTL_FILE.FRENAME (
   location  IN VARCHAR2,
   filename  IN VARCHAR2,
   dest_dir  IN VARCHAR2,
   dest_file IN VARCHAR2,
   overwrite IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 95–25 FRENAME Parameters

Parameters	Description
location (IN)	The directory location of the source file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)
filename (IN)	The source file to be renamed
dest_dir (IN)	The destination directory of the destination file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)
dest_file (N)	The new name of the file.
overwrite (IN)	The default is FALSE

96

UTL_HTTP

The UTL_HTTP package makes Hypertext Transfer Protocol (HTTP) callouts from SQL and PL/SQL. You can use it to access data on the Internet over HTTP.

With UTL_HTTP, you can write PL/SQL programs that communicate with Web (HTTP) servers. UTL_HTTP also contains a function that can be used in SQL queries. The package also supports HTTP over the Secured Socket Layer protocol (SSL), also known as HTTPS, directly or through an HTTP proxy. Other Internet-related data-access protocols (such as the File Transfer Protocol (FTP) or the Gopher protocol) are also supported using an HTTP proxy server that supports those protocols.

When the package fetches data from a Web site using HTTPS, it requires Oracle Wallet Manager to set up an Oracle wallet. Non-HTTPS fetches do not require an Oracle wallet.

See Also:

- Chapter 102, "UTL_URL"
- Chapter 100, "UTL_SMTP"
- *Oracle Advanced Security Administrator's Guide* for more information on Wallet Manager

This chapter discusses the following topics:

UTL_HTTP Constants, Types and Flow

UTL_HTTP Constants

Table 96–1 UTL_HTTP Constants

Constant and Syntax	Purpose
`HTTP_VERSION_1_0 CONSTANT VARCHAR2(10) := 'HTTP/1.0';`	Denotes HTTP version 1.0 that can be used in the function `begin_request`.
`HTTP_VERSION_1 CONSTANT VARCHAR2(10) := 'HTTP/1.1';`	Denotes HTTP version 1.1 that can be used in the function `begin_request`.
`DEFAULT_HTTP_PORT CONSTANT PLS_INTEGER := 80;`	The default TCP / IP port (80) at which a Web server or proxy server listens
`DEFAULT_HTTPS_PORT CONSTANT PLS_INTEGER := 443;`	The default TCP / IP port (443) at which an HTTPS Web server listens

The following denote all the HTTP 1.1 status codes:

```
HTTP_CONTINUE CONSTANT PLS_INTEGER := 100;          -
HTTP_SWITCHING_PROTOCOLS CONSTANT PLS_INTEGER := 101;
HTTP_OK CONSTANT PLS_INTEGER := 200;                -
HTTP_CREATED CONSTANT PLS_INTEGER := 201;           -
HTTP_ACCEPTED CONSTANT PLS_INTEGER := 202;          -
HTTP_NON_AUTHORITATIVE_INFO CONSTANT PLS_INTEGER := 203;
HTTP_NO_CONTENT CONSTANT PLS_INTEGER := 204;        -
HTTP_RESET_CONTENT CONSTANT PLS_INTEGER := 205;     -
HTTP_PARTIAL_CONTENT CONSTANT PLS_INTEGER := 206;   -
HTTP_MULTIPLE_CHOICES CONSTANT PLS_INTEGER := 300; -
HTTP_MOVED_PERMANENTLY CONSTANT PLS_INTEGER := 301;
HTTP_FOUND CONSTANT PLS_INTEGER := 302;             -
HTTP_SEE_OTHER CONSTANT PLS_INTEGER := 303;         -
HTTP_NOT_MODIFIED CONSTANT PLS_INTEGER := 304;      -
HTTP_USE_PROXY CONSTANT PLS_INTEGER := 305;         -
HTTP_TEMPORARY_REDIRECT CONSTANT PLS_INTEGER := 307;
```

Table 96–1 UTL_HTTP Constants

Constant and Syntax	Purpose
`HTTP_BAD_REQUEST CONSTANT PLS_INTEGER := 400;`	-
`HTTP_UNAUTHORIZED CONSTANT PLS_INTEGER := 401;`	-
`HTTP_PAYMENT_REQUIRED CONSTANT PLS_INTEGER := 402;`	-
`HTTP_FORBIDDEN CONSTANT PLS_INTEGER := 403;`	-
`HTTP_NOT_FOUND CONSTANT PLS_INTEGER := 404;`	-
`HTTP_NOT_ACCEPTABLE CONSTANT PLS_INTEGER := 406;`	-
`HTTP_PROXY_AUTH_REQUIRED CONSTANT PLS_INTEGER := 407;`	
`HTTP_REQUEST_TIME_OUT CONSTANT PLS_INTEGER := 408;`	-
`HTTP_CONFLICT CONSTANT PLS_INTEGER := 409;`	-
`HTTP_GONE CONSTANT PLS_INTEGER := 410;`	-
`HTTP_LENGTH_REQUIRED CONSTANT PLS_INTEGER := 411;`	-
`HTTP_PRECONDITION_FAILED CONSTANT PLS_INTEGER := 412;`	
`HTTP_REQUEST_ENTITY_TOO_LARGE CONSTANT PLS_INTEGER := 413;`	
`HTTP_REQUEST_URI_TOO_LARGE CONSTANT PLS_INTEGER := 414;`	
`HTTP_UNSUPPORTED_MEDIA_TYPE CONSTANT PLS_INTEGER := 415;`	
`HTTP_REQ_RANGE_NOT_SATISFIABLE CONSTANT PLS_INTEGER := 416;`	
`HTTP_EXPECTATION_FAILED CONSTANT PLS_INTEGER := 417;`	
`HTTP_NOT_IMPLEMENTED CONSTANT PLS_INTEGER := 501;`	-
`HTTP_BAD_GATEWAY CONSTANT PLS_INTEGER := 502;`	-
`HTTP_SERVICE_UNAVAILABLE CONSTANT PLS_INTEGER := 503;`	
`HTTP_GATEWAY_TIME_OUT CONSTANT PLS_INTEGER := 504;`	-
`HTTP_VERSION_NOT_SUPPORTED CONSTANT PLS_INTEGER := 505;`	-

UTL_HTTP Types

Use the following types with UTL_HTTP.

REQ Type

Use this PL/SQL record type to represent an HTTP request.

Syntax

```
TYPE req IS RECORD (
   url          VARCHAR2(32767),
   method       VARCHAR2(64),
   http_version  VARCHAR2(64),
);
```

Parameters

Table 96–2 REQ Type Parameters

Parameter	Description
`url`	The URL of the HTTP request. It is set after the request is created by `begin_request`.
`method`	The method to be performed on the resource identified by the URL. It is set after the request is created by `begin_request`.
`http_version`	The HTTP protocol version used to send the request. It is set after the request is created by `begin_request`.

Usage Notes

The information returned in `REQ` from the API `begin_request` is for read only. Changing the field values in the record has no effect on the request.

There are other fields in `REQ` record type whose names begin with the prefix `private_`. The fields are private and are intended for use by implementation of the `UTL_HTTP` package. You should not modify the fields.

RESP Type

This PL/SQL record type is used to represent an HTTP response.

Syntax

```
TYPE resp IS RECORD (
   status_code    PLS_INTEGER,
   reason_phrase  VARCHAR2(256),
   http_version   VARCHAR2(64),
);
```

Parameters

Table 96–3 RESP Type Parameters

Parameter	Description
status_code	The status code returned by the Web server. It is a 3-digit integer that indicates the results of the HTTP request as handled by the Web server. It is set after the response is processed by get_response.
reason_phrase	The short textual message returned by the Web server that describe the status code. It gives a brief description of the results of the HTTP request as handled by the Web server. It is set after the response is processed by get_response.
http_version	The HTTP protocol version used in the HTTP response. It is set after the response is processed by get_response.

Usage Notes

The information returned in RESP from the API get_response is read-only. There are other fields in the RESP record type whose names begin with the prefix private_. The fields are private and are intended for use by implementation of the UTL_HTTP package. You should not modify the fields.

COOKIE and COOKIE_TABLE Types

The COOKIE type is the PL/SQL record type that represents an HTTP cookie. The COOKIE_TABLE type is a PL/SQL index-by-table type that represents a collection of HTTP cookies.

Syntax

```
TYPE cookie IS RECORD (
   name  VARCHAR2(256),
   value  VARCHAR2(1024),
   domain  VARCHAR2(256),
   expire  TIMESTAMP WITH TIME ZONE,
   path  VARCHAR2(1024),
   secure  BOOLEAN,
   version  PLS_INTEGER,
   comment  VARCHAR2(1024)
);
TYPE cookie_table IS TABLE OF cookie INDEX BY binary_integer;
```

Fields of COOKIE Record Type

Table 96–4 shows the fields for the COOKIE and COOKIE_TABLE record types.

Table 96–4 Fields of COOKIE and COOKIE_TABLE Type

Field	Description
name	The name of the HTTP cookie
value	The value of the cookie
domain	The domain for which the cookie is valid
expire	The time by which the cookie will expire
path	The subset of URLs to which the cookie applies
secure	Should the cookie be returned to the Web server using secured means only.
version	The version of the HTTP cookie specification the cookie conforms. This field is NULL for Netscape cookies.
comment	The comment that describes the intended use of the cookie. This field is NULL for Netscape cookies.

Usage Notes

PL/SQL programs do not usually examine or change the cookie information stored in the UTL_HTTP package. The cookies are maintained by the package transparently. They are maintained inside the UTL_HTTP package, and they last for the duration of the database session only. PL/SQL applications that require cookies to be maintained beyond the lifetime of a database session can read the cookies using get_cookies, store them persistently in a database table, and re-store the cookies back in the package using add_cookies in the next database session. All the fields in the cookie record, except for the comment field, must be stored. Do not alter the cookie information, which can result in an application error in the Web server or compromise the security of the PL/SQL and the Web server applications. See "Example: Retrieving and Restoring Cookies" on page 96-14.

CONNECTION Type

Use this PL/SQL record type to represent the remote hosts and TCP/IP ports of a network connection that is kept persistent after an HTTP request is completed, according to the HTTP 1.1 protocol specification. The persistent network connection may be reused by a subsequent HTTP request to the same host and port. The subsequent HTTP request may be completed faster because the network connection latency is avoided. connection_table is a PL/SQL table of connection.

For a direct HTTP persistent connection to a Web server, the `host` and `port` fields contain the host name and TCP/IP port number of the Web server. The `proxy_host` and `proxy_port` fields are not set. For an HTTP persistent connection that was previously used to connect to a Web server using a proxy, the `proxy_host` and `proxy_port` fields contain the host name and TCP/IP port number of the proxy server. The host and port fields are not set, which indicates that the persistent connection, while connected to a proxy server, is not bound to any particular target Web server. An HTTP persistent connection to a proxy server can be used to access any target Web server that is using a proxy.

The `ssl` field indicates if Secured Socket Layer (SSL) is being used in an HTTP persistent connection. An HTTPS request is an HTTP request made over SSL. For an HTTPS (SSL) persistent connection connected using a proxy, the host and port fields contain the host name and TCP/IP port number of the target HTTPS Web server and the fields will always be set. An HTTPS persistent connection to an HTTPS Web server using a proxy server can only be reused to make another request to the same target Web server.

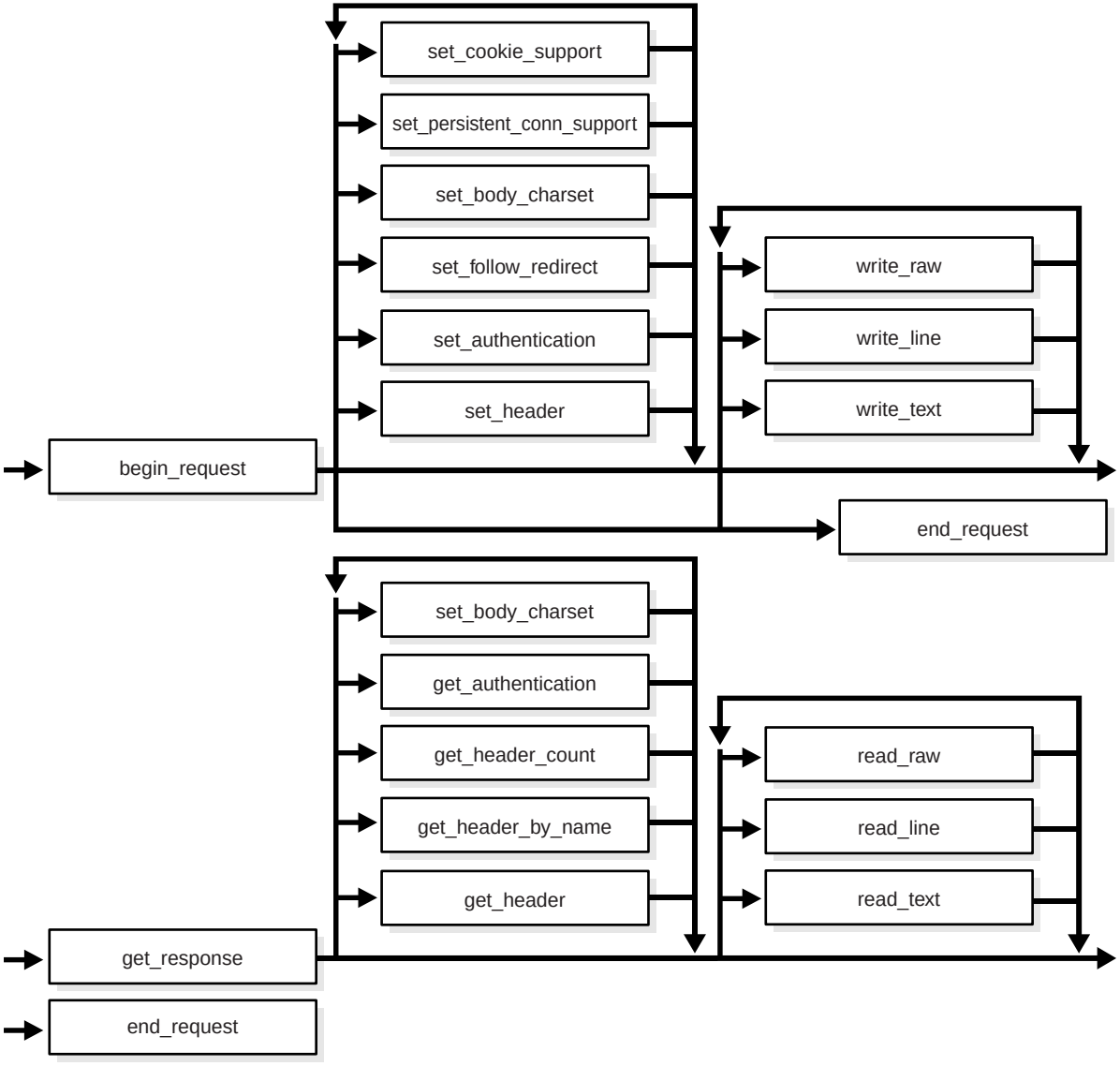
Syntax

```
TYPE connection IS RECORD (
   host  VARCHAR2(256),
   port  PLS_INTEGER,
   proxy_host  VARCHAR2(256),
   proxy_port  PLS_INTEGER,
   ssl  BOOLEAN
);
TYPE connection_table IS TABLE OF connection INDEX BY BINARY_INTEGER;
```

UTL_HTTP Flow

The `UTL_HTTP` package provides access to the HTTP protocol. The API must be called in the order shown in Figure 96–1, or an exception will be raised.

Figure 96–1 Flow of the Core UTL_HTTP Package



The following can be called at any time:

- Non-protocol APIs that manipulate cookies
  - `get_cookie_count`
  - `get_cookies`
  - `add_cookies`
  - `clear_cookies`
- Persistent connections
  - `get_persistent_conn_count`
  - `get_persistent_conns`
  - `close_persistent_conn`
  - `close_persistent_conns`
- APIs that manipulate attributes and configurations of the `UTL_HTTP` package in the current session
  - `set_proxy`
  - `get_proxy`
  - `set_cookie_support`
  - `get_cookie_support`
  - `set_follow_redirect`
  - `get_follow_redirect`
  - `set_body_charset`
  - `get_body_charset`
  - `set_persistent_conn_support`
  - `get_persistent_conn_support`
  - `set_detailed_excp_support`
  - `get_detailed_excp_support`
  - `set_wallet`
  - `set_transfer_timeout`
  - `get_transfer_timeout`

- APIs that retrieve the last detailed exception code and message UTL_HTTP package in the current session
  - get_detailed_sqlcode
  - get_detailed_sqlerrm

NOTE: Some of the request and response APIs bear the same name as the API that manipulates the attributes and configurations of the package in the current session. They are overloaded versions of the API that manipulate a request or a response.

UTL_HTTP Exceptions

Table 96–5 lists the exceptions that the UTL_HTTP package API can raise. By default, UTL_HTTP raises the exception request_failed when a request fails to execute. If the package is set to raise a detailed exception by set_detailed_excp_support, the rest of the exceptions will be raised directly (except for the exception end_of_body, which will be raised by read_text, read_line, and read_raw regardless of the setting).

Table 96–5 UTL_HTTP Exceptions

Exception	Error Code	Reason	Where Raised
request_failed	29273	The request fails to executes	Any HTTP request or response API when detailed_exception is disabled
bad_argument	29261	The argument passed to the API is bad	Any HTTP request or response API when detailed_exception is enabled
bad_url	29262	The requested URL is badly formed	begin_request, when detailed_exception is enabled
protocol_error	29263	An HTTP protocol error occurs when communicating with the Web server	set_header, get_response, read_raw, read_text, and read_line, when detailed_exception is enabled
unknown_scheme	29264	The scheme of the requested URL is unknown	begin_request and get_response, when detailed_exception is enabled

Table 96–5 UTL_HTTP Exceptions

Exception	Error Code	Reason	Where Raised
header_not_found	29265	The header is not found	get_header, get_header_by_name, when detailed_exception is enabled
end_of_body	29266	The end of HTTP response body is reached	read_raw, read_text, and read_line, when detailed_exception is enabled
illegal_call	29267	The call to UTL_HTTP is illegal at the current state of the HTTP request	set_header, set_authentication, and set_persistent_conn_support, when detailed_exception is enabled
http_client_error	29268	From get_response, the response status code indicates that a client error has occurred (status code in 4xx range). Or from begin_request, the HTTP proxy returns a status code in the 4xx range when making an HTTPS request through the proxy.	get_response, begin_request when detailed_exception is enabled
http_server_error	29269	From get_response, the response status code indicates that a client error has occurred (status code in 5xx range). Or from begin_request, the HTTP proxy returns a status code in the 5xx range when making an HTTPS request through the proxy.	get_response, begin_request when detailed_exception is enabled
too_many_requests	29270	Too many requests or responses are open	begin_request, when detailed_exception is enabled
partial_multibyte_exception	29275	No complete character is read and a partial multibyte character is found at the end of the response body	read_text and read_line, when detailed_exception is enabled
transfer_timeout	29276	No data is read and a read timeout occurred	read_text and read_line, when detailed_exception is enabled

NOTE: The `partial_multibyte_char` and `transfer_timeout` exceptions are duplicates of the same exceptions defined in `UTL_TCP`. They are defined in this package so that the use of this package does not require the knowledge of the `UTL_TCP`. As those exceptions are duplicates, an exception handle that catches the `partial_multibyte_char` and `transfer_timeout` exceptions in this package also catch the exceptions in the `UTL_TCP`.

For REQUEST and REQUEST_PIECES(), the `request_failed` exception is raised when any exception occurs and `detailed_exception` is disabled.

UTL_HTTP Examples

The following examples demonstrate how to use UTL_HTTP.

Example: Using UTL_HTTP

```
SET serveroutput ON SIZE 40000

DECLARE
  req   utl_http.req;
  resp  utl_http.resp;
  value VARCHAR2(1024);
BEGIN

  utl_http.set_proxy('proxy.my-company.com', 'corp.my-company.com');

  req := utl_http.begin_request('http://www-hr.corp.my-company.com');
  utl_http.set_header(req, 'User-Agent', 'Mozilla/4.0');
  resp := utl_http.get_response(req);
  LOOP
    utl_http.read_line(resp, value, TRUE);
    dbms_output.put_line(value);
  END LOOP;
  utl_http.end_response(resp);
EXCEPTION
  WHEN utl_http.end_of_body THEN
    utl_http.end_response(resp);
END;
```

Example: Retrieving HTTP Response Headers

```
SET serveroutput ON SIZE 40000

DECLARE
  req   utl_http.req;
  resp  utl_http.resp;
  name  VARCHAR2(256);
  value VARCHAR2(1024);
BEGIN

  utl_http.set_proxy('proxy.my-company.com', 'corp.my-company.com');

  req := utl_http.begin_request('http://www-hr.corp.my-company.com');
  utl_http.set_header(req, 'User-Agent', 'Mozilla/4.0');
  resp := utl_http.get_response(req);

  dbms_output.put_line('HTTP response status code: ' || resp.status_code);
  dbms_output.put_line('HTTP response reason phrase: ' || resp.reason_phrase);

  FOR i IN 1..utl_http.get_header_count(resp) LOOP
    utl_http.get_header(resp, i, name, value);
    dbms_output.put_line(name || ': ' || value);
  END LOOP;
  utl_http.end_response(resp);
END;
```

Example: Handling HTTP Authentication

```
SET serveroutput ON SIZE 40000

CREATE OR REPLACE PROCEDURE get_page (url      IN VARCHAR2,
                                      username IN VARCHAR2 DEFAULT NULL,
                                      password IN VARCHAR2 DEFAULT NULL,
                                      realm    IN VARCHAR2 DEFAULT NULL) AS
  req       utl_http.req;
  resp      utl_http.resp;
  my_scheme VARCHAR2(256);
  my_realm  VARCHAR2(256);
  my_proxy  BOOLEAN;
BEGIN

  -- Turn off checking of status code. We will check it by ourselves.
  utl_http.http_response_error_check(FALSE);

  req := utl_http.begin_request(url);
```

```
  IF (username IS NOT NULL) THEN
    utl_http.set_authentication(req, username, password); -- Use HTTP Basic
Authen. Scheme
  END IF;

  resp := utl_http.get_response(req);
  IF (resp.status_code = utl_http.HTTP_UNAUTHORIZED) THEN
    utl_http.get_authentication(resp, my_scheme, my_realm, my_proxy);
    IF (my_proxy) THEN
       dbms_output.put_line('Web proxy server is protected.');
       dbms_output.put('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the proxy
server.');
    ELSE
      dbms_output.put_line('Web page ' || url || ' is protected.');
      dbms_output.put('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the Web
page.');
    END IF;
    utl_http.end_response(resp);
    RETURN;
  END IF;

  FOR i IN 1..utl_http.get_header_count(resp) LOOP
    utl_http.get_header(resp, i, name, value);
    dbms_output.put_line(name || ': ' || value);
  END LOOP;
  utl_http.end_response(resp);

END;
```

Example: Retrieving and Restoring Cookies

```
CREATE TABLE my_cookies (
    session_id  BINARY_INTEGER,
    name        VARCHAR2(256),
    value       VARCHAR2(1024),
    domain      VARCHAR2(256),
    expire      DATE,
    path        VARCHAR2(1024),
    secure      VARCHAR2(1),
    version     BINARY_INTEGER
);

CREATE SEQUENCE session_id;
```

```
SET serveroutput ON SIZE 40000

REM Retrieve cookies from UTL_HTTP

CREATE OR REPLACE FUNCTION save_cookies RETURN BINARY_INTEGER AS
  cookies        utl_http.cookie_table;
  my_session_id  BINARY_INTEGER;
  secure         VARCHAR2(1);
BEGIN

  /* assume that some cookies have been set in previous HTTP requests. */

  utl_http.get_cookies(cookies);
  select session_id.nextval into my_session_id from dual;

  FOR i in 1..cookies.count LOOP
    IF (cookies(i).secure) THEN
      secure := 'Y';
    ELSE
      secure := 'N';
    END IF;
    insert into my_cookies
    value (my_session_id, cookies(i).name, cookies(i).value, cookies(i).domain,
           cookies(i).expire, cookies(i).path, secure, cookies(i).version);
  END LOOP;

  RETURN my_session_id;

END;

REM Retrieve cookies from UTL_HTTP

CREATE OR REPLACE PROCEDURE restore_cookies (this_session_id IN BINARY_INTEGER)
AS
  cookies        utl_http.cookie_table;
  cookie         utl_http.cookie;
  i              PLS_INTEGER := 0;
  CORSOR c (c_session_id BINARY_INTEGER) IS
    SELECT * FROM my_cookies WHERE session_id = c_session_id;
BEGIN

  FOR r IN c(this_session_id) LOOP
    i := i + 1;
    cookie.name   := r.name;
```

```
    cookie.value  := r.value;
    cookie.domain := r.domain;
    cookie.expire := r.expire;
    cookie.path   := r.path;
    IF (r.secure = 'Y') THEN
      cookie.secure := TRUE;
    ELSE
      cookie.secure := FALSE;
    END IF;
    cookie.version := r.version;
    cookies(i) := cookie;
  END LOOP;

  utl_http.clear_cookies;
  utl_http.add_cookies(cookies);

END;
```

Summary of UTL_HTTP Subprograms

Table 96–6 UTL_HTTP Subprograms—Simple HTTP Fetches in a Single Call

Subprogram	Description
REQUEST Function on page 96-21	Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.
REQUEST_PIECES Function on page 96-23	Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.

Table 96–7 UTL_HTTP Subprograms—Session Settings

Subprogram	Description
SET_PROXY Procedure on page 96-26	Sets the proxy to be used for requests of HTTP or other protocols
GET_PROXY Procedure on page 96-27	Retrieves the current proxy settings
SET_COOKIE_SUPPORT Procedure on page 96-28	Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session
GET_COOKIE_SUPPORT Procedure on page 96-29	Retrieves the current cookie support settings

Table 96–7 (Cont.) UTL_HTTP Subprograms—Session Settings

Subprogram	Description
SET_FOLLOW_REDIRECT Procedure on page 96-30	Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the get_response function
GET_FOLLOW_REDIRECT Procedure on page 96-31	Retrieves the follow-redirect setting in the current session
SET_BODY_CHARSET Procedure on page 96-31	Sets the default character set of the body of all future HTTP requests when the media type is text and the character set is not specified in the Content-Type header
GET_BODY_CHARSET Procedure on page 96-32	Retrieves the default character set of the body of all future HTTP requests
SET_PERSISTENT_CONN_SUPPORT Procedure on page 96-32	Sets whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session
GET_PERSISTENT_CONN_SUPPORT Procedure on page 96-35	Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session
SET_RESPONSE_ERROR_CHECK Procedure on page 96-35	Sets whether or not get_response raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges
GET_RESPONSE_ERROR_CHECK Procedure on page 96-36	Checks if the response error check is set or not
SET_DETAILED_EXCP_SUPPORT Procedure on page 96-37	Sets the UTL_HTTP package to raise a detailed exception
GET_DETAILED_EXCP_SUPPORT Procedure on page 96-37	Checks if the UTL_HTTP package will raise a detailed exception or not
SET_WALLET Procedure on page 96-37	Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS
SET_TRANSFER_TIMEOUT Procedure on page 96-39	Sets the timeout value for UTL_HTTP to read the HTTP response from the Web server or proxy server
GET_TRANSFER_TIMEOUT Procedure on page 96-39	Retrieves the current network transfer timeout value

Table 96–8 UTL_HTTP Subprograms—HTTP Requests

Subprogram	Description
BEGIN_REQUEST Function on page 96-40	Begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line.
SET_HEADER Procedure on page 96-41	Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.
SET_AUTHENTICATION Procedure on page 96-42	Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.
SET_COOKIE_SUPPPORT Procedure on page 96-42	Enables or disables support for the HTTP cookies in the request.
SET_FOLLOW_REDIRECT Procedure on page 96-43	Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE function.
SET_BODY_CHARSET Procedure on page 96-44	Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header.
SET_PERSISTENT_CONN_SUPPORT Procedure on page 96-45	Enables or disables support for the HTTP 1.1 persistent-connection in the request.
WRITE_TEXT Procedure on page 96-47	Writes some text data in the HTTP request body.
WRITE_LINE Procedure on page 96-48	Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP).
WRITE_RAW Procedure on page 96-50	Writes some binary data in the HTTP request body.
END_REQUEST Procedure on page 96-50	Ends the HTTP request.

Table 96–9 UTL_HTTP Subprograms—HTTP Responses

Subprogram	Description
GET_RESPONSE Function on page 96-51	Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed.
GET_HEADER_COUNT Function on page 96-52	Returns the number of HTTP response headers returned in the response.
GET_HEADER Procedure on page 96-52	Returns the n^{th} HTTP response header name and value returned in the response.

Table 96–9 (Cont.) UTL_HTTP Subprograms—HTTP Responses

Subprogram	Description
GET_HEADER_BY_NAME Procedure on page 96-53	Returns the HTTP response header value returned in the response given the name of the header.
GET_AUTHENTICATION Procedure on page 96-54	Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.
SET_BODY_CHARSET Procedure on page 96-55	Sets the character set of the response body when the media type is "text" but the character set is not specified in the "Content-Type" header.
READ_TEXT Procedure on page 96-56	Reads the HTTP response body in text form and returns the output in the caller-supplied buffer.
READ_LINE Procedure on page 96-57	Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer.
READ_RAW Procedure on page 96-58	Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer.
END_RESPONSE Procedure on page 96-59	Ends the HTTP response. It completes the HTTP request and response.

Table 96–10 UTL_HTTP Subprograms—HTTP Cookies

Subprogram	Description
GET_COOKIE_COUNT Function on page 96-60	Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers.
GET_COOKIES Function on page 96-60	Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers.
ADD_COOKIES Procedure on page 96-60	Adds the cookies maintained by UTL_HTTP.
CLEAR_COOKIES Procedure on page 96-61	Clears all cookies maintained by the UTL_HTTP package.

Table 96–11 UTL_HTTP Subprograms—HTTP Persistent Connections

Subprogram	Description
GET_PERSISTENT_CONN_COUNT Function on page 96-61	Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers.
GET_PERSISTENT_CONNS Procedure on page 96-62	Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers.
CLOSE_PERSISTENT_CONN Procedure on page 96-62	Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session.
CLOSE_PERSISTENT_CONNS Procedure on page 96-63	Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session.

Table 96–12 UTL_HTTP Subprograms—Error Conditions

Subprogram	Description
GET_DETAILED_SQLCODE Function on page 96-64	Retrieves the detailed SQLCODE of the last exception raised.
GET_DETAILED_SQLERRM Function on page 96-65	Retrieves the detailed SQLERRM of the last exception raised.

Simple HTTP Fetches

`REQUEST` and `REQUEST_PIECES` take a string universal resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

You should not expect `REQUEST` or `REQUEST_PIECES` to succeed in contacting a URL unless you can contact that URL by using a browser on the same machine (and with the same privileges, environment variables, and so on.)

If `REQUEST` or `REQUEST_PIECES` fails (for example, if it raises an exception, or if it returns an HTML-formatted error message, but you believe that the URL argument is correct), then try contacting that same URL with a browser to verify network availability from your machine. You may have a proxy server set in your browser that needs to be set with each `REQUEST` or `REQUEST_PIECES` call using the optional `proxy` parameter.

Note: UTL_HTTP can also use environment variables to specify its proxy behavior. For example, on UNIX, setting the environment variable http_proxy to a URL uses that service as the proxy server for HTTP requests. Setting the environment variable no_proxy to a domain name does not use the HTTP proxy server for URLs in that domain. When the UTL_HTTP package is executed in the Oracle database server, the environment variables are the ones that are set when the database instance is started.

REQUEST Function

This function returns up to the first 2000 bytes of data retrieved from the given URL. This function can be used directly in SQL queries.

Syntax

```
UTL_HTTP.REQUEST (
   url             IN VARCHAR2,
   proxy           IN VARCHAR2 DEFAULT NULL),
   wallet_path     IN VARCHAR2 DEFAULT NULL,
   wallet_password IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references (request, wnds, rnds, wnps, rnps);
```

Parameters

Table 96–13 REQUEST Function Parameters

Parameter	Description
url	Universal resource locator.
proxy	(Optional) Specifies a proxy server to use when making the HTTP request. See set_proxy for the full format of the proxy setting.

Table 96–13 REQUEST Function Parameters

Parameter	Description
wallet_path	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\<username>\WALLETS, and in Unix is, for example, file:/home/<username>/wallets When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server. See set_wallet for a description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
wallet_password	(Optional) Specifies the password required to open the wallet.

Returns

The return type is a string of length 2000 or less, which contains up to the first 2000 bytes of the HTML result returned from the HTTP request to the argument URL.

Exceptions

INIT_FAILED
REQUEST_FAILED

Usage Notes

The URL passed as an argument to this function is not examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Please see the documentation of the function set_wallet on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

<HTML>
<HEAD>

```
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

Example

```
SQLPLUS> SELECT utl_http.request('http://www.my-company.com/') FROM dual;
UTL_HTTP.REQUEST('HTTP://WWW.MY-COMPANY.COM/')
<html>
<head><title>My Company Home Page</title>
<!--changed Jan. 16, 19
1 row selected.
```

If you are behind a firewall, include the `proxy` parameter. For example, from within the Oracle firewall, where there might be a proxy server named `www-proxy.my-company.com`:

```
SQLPLUS> SELECT
utl_http.request('http://www.my-company.com', 'www-proxy.us.my-company.com')
FROM dual;
```

REQUEST_PIECES Function

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.

Syntax

```
type html_pieces is table of varchar2(2000) index by binary_integer;

UTL_HTTP.REQUEST_PIECES (
   url             IN VARCHAR2,
   max_pieces      IN NATURAL DEFAULT 32767,
   proxy           IN VARCHAR2 DEFAULT NULL,
   wallet_path     IN VARCHAR2 DEFAULT NULL,
```

```
   wallet_password IN VARCHAR2 DEFAULT NULL)
RETURN html_pieces;
```

Pragmas

```
pragma restrict_references (request_pieces, wnds, rnds, wnps, rnps);
```

Parameters

Table 96–14 REQUEST_PIECES Function Parameters

Parameter	Description
`url`	Universal resource locator.
`max_pieces`	(Optional) The maximum number of pieces (each 2000 characters in length, except for the last, which may be shorter), that `REQUEST_PIECES` should return. If provided, then that argument should be a positive integer.
`proxy`	(Optional) Specifies a proxy server to use when making the HTTP request. See `set_proxy` for the full format of the proxy setting.
`wallet_path`	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of wallet_path is 'file:/<local-dir-for-client-side-wallet>'. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\<username>\WALLETS, and in Unix is, for example, file:/home/<username>/wallets. When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server. See `set_wallet` for the description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
`wallet_password`	(Optional) Specifies the password required to open the wallet.

Returns

`REQUEST_PIECES` returns a PL/SQL table of type `UTL_HTTP.HTML_PIECES`. Each element of that PL/SQL table is a string of maximum length 2000. The elements of the PL/SQL table returned by `REQUEST_PIECES` are successive pieces of the data obtained from the HTTP request to that URL.

Exceptions

```
INIT_FAILED
REQUEST_FAILED
```

Usage Notes

The URL passed as an argument to this function will not be examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Each entry of the PL/SQL table (the "pieces") returned by this function may not be filled to their fullest capacity. The function may start filling the data in the next piece before the previous "piece" is totally full.

Please see the documentation of the function set_wallet on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

Example

```
SET SERVEROUTPUT ON
```

```
DECLARE
   x   utl_http.html_pieces;
   len PLS_INTEGER;
BEGIN
   x := utl_http.request_pieces('http://www.oracle.com/', 100);
   dbms_output.put_line(x.count || ' pieces were retrieved.');
   dbms_output.put_line('with total length ');
   IF x.count < 1 THEN
      dbms_output.put_line('0');
  ELSE
   len := 0;
   FOR i in 1..x.count LOOP
      len := len + length(x(i));
   END LOOP;
   dbms_output.put_line(i);
  END IF;
END;
/
-- Output
Statement processed.
4 pieces were retrieved.
with total length
7687
```

Session Settings

Session settings manipulate the configuration and default behavior of UTL_HTTP when HTTP requests are executed within a database user session. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. Those settings can be changed later by calling the request API. When a response is created for a request, it inherits those settings from the request. Only the body character set can be changed later by calling the response API.

SET_PROXY Procedure

This procedure sets the proxy to be used for requests of the HTTP or other protocols, excluding those for hosts that belong to the domain specified in no_proxy_domains. The proxy may include an optional TCP/IP port number at which the proxy server listens. The syntax is [http://]host[:port][/], for example, www-proxy.my-company.com:80. If the port is not specified for the proxy, port 80 is assumed. no_proxy_domains is a comma-, semi-colon-, or space-separated list of domains or hosts for which HTTP requests should be sent

directly to the destination HTTP server instead of going through a proxy server. Optionally, a port number can be specified for each domain or host. If the port number is specified, the no-proxy restriction is only applied to the request at the port of the particular domain or host, for example, `corp.my-company.com, eng.my-company.com:80`. When `no_proxy_domains` is `NULL` and the proxy is set, all requests go through the proxy. When the proxy is not set, `UTL_HTTP` sends requests to the target Web servers directly.

Syntax

```
UTL_HTTP.set_proxy (
   proxy            IN VARCHAR2,
   no_proxy_domains  IN VARCHAR2);
```

Parameters

Table 96–15 SET_PROXY Procedure Parameters

Parameter	Description
`proxy (IN)`	The proxy (host and an optional port number) to be used by the UTL_HTTP package
`no_proxy_domains (IN)`	The list of hosts and domains for which no proxy should be used for all requests.

Usage Notes

If proxy settings are set when the database server instance is started, the proxy settings in the environment variables `http_proxy` and `no_proxy` are assumed. Proxy settings set by this procedure override the initial settings.

GET_PROXY Procedure

This procedure retrieves the current proxy settings.

Syntax

```
UTL_HTTP.get_proxy (
   proxy            OUT NOCOPY VARCHAR2,
   no_proxy_domains  OUT NOCOPY VARCHAR2);
```

Parameters

Table 96–16 GET_PROXY Procedure Parameters

Parameter	Description
proxy (OUT)	The proxy (host and an optional port number) currently used by the UTL_HTTP package
no_proxy_domains (OUT)	The list of hosts and domains for which no proxy is used for all requests.

SET_COOKIE_SUPPORT Procedure

This procedure sets:

- Whether or not future HTTP requests will support HTTP cookies
- The maximum number of cookies maintained in the current database user session

If cookie support is enabled for an HTTP request, all cookies saved in the current session and applicable to the request are returned to the Web server in the request, in accordance with HTTP cookie specification standards. Cookies that are set in response to the request are saved in the current session for return to the Web server in subsequent requests, if cookie support is enabled for those requests. If cookie support is disabled for an HTTP request, no cookies will be returned to the Web server in the request and the cookies set in the response to the request are not saved in the current session, although the `Set-Cookie` HTTP headers can still be retrieved from the response.

Cookie support is enabled by default for all HTTP requests in a database user session. The default setting of the cookie support (enabled versus disabled) affects only the future requests and has no effect on the existing ones. After your request is created, the cookie support setting may be changed by using the other `set_cookie_support` procedure that operates on a request.

The default maximum number of cookies saved in the current session is 20 for each site and 300 total.

Syntax

```
UTL_HTTP.set_cookie_support (
   enable       IN BOOLEAN,
   max_cookies  IN PLS_INTEGER DEFAULT 300,
   max_cookies_per_site  IN PLS_INTEGER DEFAULT 20);
```

Parameters

Table 96–17 SET_COOKIE SUPPORT Procedure Parameters

Parameter	Description
`enable (IN)`	Sets whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)
`max_cookies (IN)`	Sets the maximum total number of cookies maintained in the current session
`max_cookies_per_site (IN)`	Sets the maximum number of cookies maintained in the current session for each Web site

Usage Notes

If you lower the maximum total number of cookies or the maximum number of cookies for each Web site, the oldest cookies will be purged first to reduce the number of cookies to the lowered maximum. HTTP cookies saved in the current session last for the duration of the database session only; there is no persistent storage for the cookies. Cookies saved in the current session are not cleared if you disable cookie support.

See "UTL_HTTP Examples" on page 96-12 for how to use `get_cookies` and `add_cookies` to retrieve, save, and restore cookies.

GET_COOKIE_SUPPORT Procedure

This procedure retrieves the current cookie support settings.

Syntax

```
UTL_HTTP.get_cookie_support (
   enable                OUT BOOLEAN,
   max_cookies           OUT PLS_INTEGER,
   max_cookies_per_site  OUT PLS_INTEGER);
```

Parameters

Table 96–18 GET_COOKIE SUPPORT Procedure Parameters

Parameter	Description
`enable (OUT)`	Indicates whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)

Table 96–18 GET_COOKIE SUPPORT Procedure Parameters

Parameter	Description
max_cookies (OUT)	Indicates the maximum total number of cookies maintained in the current session
max_cookies_per_site (OUT)	Indicates the maximum number of cookies maintained in the current session for each Web site

SET_FOLLOW_REDIRECT Procedure

This procedure sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the get_response function.

If max_redirects is set to a positive number, get_response will automatically follow the redirected URL for the HTTP response status code 301, 302, and 307 for the HTTP HEAD and GET methods, and 303 for all HTTP methods, and retry the HTTP request (the request method will be changed to HTTP GET for the status code 303) at the new location. It follows the redirection until the final, non-redirect location is reached, or an error occurs, or the maximum number of redirections has been reached (to prevent an infinite loop). The URL and method fields in the REQ record will be updated to the last redirected URL and the method used to access the URL. Set the maximum number of redirects to zero to disable automatic redirection.

The default maximum number of redirections in a database user session is 3. The default value affects only future requests and has no effect on existing requests.

After a request is created, the maximum number of redirections can be changed by using the other set_follow_redirect procedure that operates on a request.

Syntax

```
UTL_HTTP.set_follow_redirect (
   max_redirects  IN PLS_INTEGER DEFAULT 3);
```

Parameters

Table 96–19 SET_FOLLOW_REDIRECT Procedure Parameters

Parameter	Description
max_redirects (IN)	The maximum number of redirections. Set to zero to disable redirection

Usage Notes

While it is set not to follow redirect automatically in the current session, it is possible to specify individual HTTP requests to follow redirect instructions the function `follow_redirect` and vice versa.

GET_FOLLOW_REDIRECT Procedure

This procedure retrieves the follow-redirect setting in the current session.

Syntax

```
UTL_HTTP.get_follow_redirect (
   max_redirects  OUT PLS_INTEGER);
```

Parameters

Table 96–20 GET_FOLLOW_REDIRECT Procedure Parameters

Parameter	Description
`max_redirects (OUT)`	The maximum number of redirections for all future HTTP requests.

SET_BODY_CHARSET Procedure

This procedure sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header. Following the HTTP protocol standard specification, if the media type of a request or a response is `text`, but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to `ISO-8859-1`. A response created for a request inherits the default body character set of the request instead of the body character set of the current session.

The default body character set is ISO-8859-1 in a database user session. The default body character set setting affects only future requests and has no effect on existing requests.

After a request is created, the body character set can be changed by using the other `set_body_charset` procedure that operates on a request.

Syntax

```
UTL_HTTP.set_body_charset (
```

```
   charset  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 96–21 SET_BODY_CHARSET Procedure Parameters

Parameter	Description
charset (IN)	The default character set of the request body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If charset is NULL, the database character set is assumed.

GET_BODY_CHARSET Procedure

This procedure retrieves the default character set of the body of all future HTTP requests.

Syntax

```
UTL_HTTP.get_body_charset (
   charset  OUT NOCOPY VARCHAR2);
```

Parameters

Table 96–22 GET_BODY_CHARSET Procedure Parameters

Parameter	Description
charset (OUT)	The default character set of the body of all future HTTP requests

SET_PERSISTENT_CONN_SUPPORT Procedure

This procedure sets:

- Whether or not future HTTP requests will support the HTTP 1.1 persistent connection
- The maximum number of persistent connections maintained in the current database user session

If persistent-connection support is enabled for an HTTP request, the package keeps the network connections to a Web server or the proxy server open in the package after the request is completed. A subsequent request to the same server can use the HTTP 1.1 persistent connection. With persistent connection support, subsequent

HTTP requests can be completed faster because network connection latency is avoided. If the persistent-connection support is disabled for a request, the package will send the HTTP header `Connection: close` automatically in the HTTP request and close the network connection when the request is completed. This setting has no effect on HTTP requests that follows HTTP 1.0 protocol, for which the network connections will always be closed after the requests are completed.

When a request is made, the package always attempts to reuse an existing persistent connection to the target Web server (or proxy server) if one is available. If none is available, a new network connection will be initiated. The persistent-connection support setting for a request affects only whether the network connection should be closed after a request completes.

Persistent-connection support is disabled for all HTTP requests in a database user session by default. The default maximum number of persistent connections saved in the current session is zero. The default setting of the persistent-connection support (enabled versus disabled) affects only future requests and has no effect on existing requests.

After a request is created, the persistent-connection support setting can be changed by using the other `set_persistent_conn_support` procedure that operates on a request.

While the use of persistent connections in `UTL_HTTP` can reduce the time it takes to fetch multiple Web pages from the same server, it consumes system resources (network connections) in the database server. Excessive use of persistent connections can reduce the scalability of the database server when too many network connections are kept open in the database server. Network connections should be kept open only if they will be used immediately by subsequent requests and should be closed when they are no longer needed. You should normally disable persistent connection support in the session and enable persistent connections in individual HTTP requests, as shown in "Example: Using SET_PERSISTENT_CONN_SUPPORT" on page 96-34.

Syntax

```
UTL_HTTP.set_persistent_conn_support (
   enable     IN BOOLEAN,
   max_conns  IN PLS_INTEGER DEFAULT 0);
```

Parameters

Table 96–23 SET_PERSISTENT_CONN_SUPPORT Procedure Parameters

Parameter	Description
enable (IN)	Enables (set to TRUE) or disables (set to FALSE) persistent connection support
max_conns (IN)	Sets the maximum number of persistent connections maintained in the current session.

Usage Notes

The default value of the maximum number of persistent connections in a database session is zero. To truly enable persistent connections, you must also set the maximum number of persistent connections to a positive value or no connections will be kept persistent.

Example: Using SET_PERSISTENT_CONN_SUPPORT

```
DECLARE
  TYPE vc2_table IS TABLE OF VARCHAR2(256) INDEX BY binary_integer;
  paths vc2_table;

  PROCEDURE fetch_pages(paths IN vc2_table) AS
    url_prefix VARCHAR2(256) := 'http://www.my-company.com/';
    req   utl_http.req;
    resp  utl_http.resp;
    data  VARCHAR2(1024);
  BEGIN
    FOR i IN 1..paths.count LOOP
      req := utl_http.begin_request(url_prefix || paths(i));

      -- Use persistent connection except for the last request
      IF (i < paths.count) THEN
        utl_http.set_persistent_conn_support(req, TRUE);
      END IF;

      resp := utl_http.get_response(req);

      BEGIN
        LOOP
          utl_http.read_text(resp, data);
          -- do something with the data
        END LOOP;
```

```
      EXCEPTION
        WHEN utl_http.end_of_body THEN
          NULL;
      END;
      utl_http.end_response(resp);
    END LOOP;
  END;

BEGIN
  utl_http.set_persistent_conn_support(FALSE, 1);
  paths(1) := '...';
  paths(2) := '...';
  ...
  fetch_pages(paths);
END;
```

GET_PERSISTENT_CONN_SUPPORT Procedure

This procedure checks:

- If the persistent connection support is enabled
- Gets the maximum number of persistent connections in the current session

Syntax

```
UTL_HTTP.get_persistent_conn_support (
   enable     OUT BOOLEAN,
   max_conns  OUT PLS_INTEGER);
```

Parameters

Table 96–24 GET_PERSISTENT_CONN_SUPPORT Procedure Parameters

Parameter	Description
`enable (OUT)`	TRUE if persistent connection support is enabled; otherwise FALSE
`max_conns (OUT)`	the maximum number of persistent connections maintained in the current session.

SET_RESPONSE_ERROR_CHECK Procedure

This procedure sets whether or not `get_response` raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or

5xx ranges. For example, when the requested URL is not found in the destination Web server, a 404 (document not found) response status code is returned. If the status code indicates an error—a 4xx or 5xx code—and this procedure is enabled, `get_response` will raise the `HTTP_CLIENT_ERROR` or `HTTP_SERVER_ERROR` exception. If `SET_RESPONSE_ERROR_CHECK` is set to `FALSE`, `get_response` will not raise an exception when the status code indicates an error. Response error check is turned off by default.

Syntax

```
UTL_HTTP.set_response_error_check (
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 96–25 SET_RESPONSE_ERROR_CHECK Procedure Parameters

Parameter	Description
`enable (IN)`	`TRUE` to check for response errors; otherwise `FALSE`

Usage Notes

The `get_response` function can raise other exceptions when SET_RESPONSE_ERROR_CHECK is set to `FALSE`.

GET_RESPONSE_ERROR_CHECK Procedure

This procedure checks if the response error check is set or not.

Syntax

```
UTL_HTTP.get_response_error_check (
   enable  OUT BOOLEAN);
```

Parameters

Table 96–26 GET_RESPONSE_ERROR_CHECK Procedure Parameters

Parameter	Description
`enable (OUT)`	`TRUE` if the response error check is set; otherwise `FALSE`

SET_DETAILED_EXCP_SUPPORT Procedure

This procedure sets the UTL_HTTP package to raise a detailed exception. By default, UTL_HTTP raises the request_failed exception when an HTTP request fails. Use GET_DETAILED_SQLCODE and GET_DETAILED_SQLEERM for more detailed information about the error.

Syntax

```
UTL_HTTP.set_detailed_excp_support (
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 96–27 SET_DETAILED_EXCP_SUPPORT Procedure Parameters

Parameter	Description
enable (IN)	Asks UTL_HTTP to raise a detailed exception directly if set to TRUE; otherwise FALSE

GET_DETAILED_EXCP_SUPPORT Procedure

This procedure checks if the UTL_HTTP package will raise a detailed exception or not.

Syntax

```
UTL_HTTP.get_detailed_excp_support (
   enable  OUT BOOLEAN);
```

Parameters

Table 96–28 GET_DETAILED_EXCP_SUPPORT Procedure Parameters

Parameter	Description
enable (OUT)	TRUE if UTL_HTTP raises a detailed exception; otherwise FALSE

SET_WALLET Procedure

This procedure sets the Oracle wallet used for all HTTP requests over Secured Socket Layer (SSL), namely HTTPS. When the UTL_HTTP package communicates with an HTTP server over SSL, the HTTP server presents its digital certificate,

which is signed by a certificate authority, to the UTL_HTTP package for identification purpose. The Oracle wallet contains the list of certificate authorities that are trusted by the user of the UTL_HTTP package. An Oracle wallet is required to make an HTTPS request.

To set up an Oracle wallet, use the Oracle Wallet Manager to create a wallet. In order for the HTTPS request to succeed, the certificate authority that signs the certificate of the remote HTTPS Web server must be one trust point set in the wallet. When a wallet is created, it is populated with a set of well-known certificate authorities as trust points. If the certificate authority that signs the certificate of the remote HTTPS Web server is not among the trust points, or the certificate authority has new root certificates, you should obtain the root certificate of that certificate authority and install it as a trust point in the wallet using Oracle Wallet Manager.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information on Wallet Manager

Syntax

```
UTL_HTTP.set_wallet (
   path      IN VARCHAR2,
   password  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 96–29 SET_WALLET Procedure Parameters

Parameter	Description
path (IN)	The directory path that contains the Oracle wallet. The format is file:<directory-path>. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\<username>\WALLETS, and in Unix is, for example, file:/home/<username>/wallets. When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server.
password (IN)	The password needed to open the wallet. A second copy of a wallet in a wallet directory that may be opened without a password. That second copy of the wallet is read-only. If the password is NULL, the UTL_HTTP package will open the second, read-only copy of the wallet instead.

SET_TRANSFER_TIMEOUT Procedure

Sets the default timeout value for all future HTTP requests that the UTL_HTTP package should attempt while reading the HTTP response from the Web server or proxy server. This timeout value may be used to avoid the PL/SQL programs from being blocked by busy Web servers or heavy network traffic while retrieving Web pages from the Web servers. The default value of the timeout is 60 seconds.

Syntax

```
UTL_HTTP.set_transfer_timeout (
   timeout  IN PLS_INTEGER DEFAULT 60);
```

Parameters

Table 96–30 SET_TRANSFER_TIMEOUT Procedure Parameters

Parameter	Description
TIMEOUT (IN)	The network transfer timeout value in seconds.

GET_TRANSFER_TIMEOUT Procedure

This procedure retrieves the default timeout value for all future HTTP requests.

Syntax

```
UTL_HTTP.get_transfer_timeout (
   timeout  OUT PLS_INTEGER);
```

Parameters

Table 96–31 GET_TRANSFER_TIMEOUT Procedure Parameters

Parameter	Description
TIMEOUT (OUT)	The network transfer timeout value in seconds.

HTTP Requests

The following APIs begin an HTTP request, manipulate attributes, and send the request information to the Web server. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. The settings can be changed by calling the request API.

BEGIN_REQUEST Function

This functions begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line. The PL/SQL program continues the request by calling some other API to complete the request.

Syntax

```
UTL_HTTP.begin_request (
   url          IN VARCHAR2,
   method       IN VARCHAR2 DEFAULT 'GET',
   http_version IN VARCHAR2 DEFAULT NULL)
RETURN req;
```

Parameters

Table 96–32 BEGIN_REQUEST Function Parameters

Parameter	Description
url (IN)	The URL of the HTTP request
method (IN)	The method performed on the resource identified by the URL
http_version (IN)	The HTTP protocol version that sends the request. The format of the protocol version is HTTP/major-version.minor-version, where major-version and minor-version are positive numbers. If this parameter is set to NULL, UTL_HTTP uses the latest HTTP protocol version that it supports to send the request. The latest version that the package supports is 1.1 and it can be upgraded to a later version. The default is NULL.

Usage Notes

The URL passed as an argument to this function is not examined for illegal characters, such as spaces, according to URL specification RFC 2396. You should escape those characters with the UTL_URL package to return illegal and reserved characters. URLs should consist of US-ASCII characters only. See Chapter 102, "UTL_URL" for a list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

An Oracle wallet must be set before accessing Web servers over HTTPS. See the set_wallet procedure on how to set up an Oracle wallet.

SET_HEADER Procedure

This procedure sets an HTTP request header. The request header is sent to the Web server as soon as it is set.

Syntax

```
UTL_HTTP.set_header (
   r      IN OUT NOCOPY req,
   name   IN VARCHAR2,
   value  IN VARCHAR2);
```

Parameters

Table 96–33 SET_HEADER Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
name (IN)	The name of the HTTP request header
value (IN)	The value of the HTTP request header

Usage Notes

Multiple HTTP headers with the same name are allowed in the HTTP protocol standard. Therefore, setting a header does not replace a prior header with the same name.

If the request is made using HTTP 1.1, UTL_HTTP sets the Host header automatically for you.

When you set the Content-Type header with this procedure, UTL_HTTP looks for the character set information in the header value. If the character set information is present, it is set as the character set of the request body. It can be overridden later by using the set_body_charset procedure.

When you set the Transfer-Encoding header with the value chunked, UTL_HTTP automatically encodes the request body written by the write_text, write_line and write_raw procedures. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format.

SET_AUTHENTICATION Procedure

This procedure sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.

Syntax

```
UTL_HTTP.set_authentication(
   r  IN OUT NOCOPY req,
   username  IN VARCHAR2,
   password  IN VARCHAR2,
   scheme    IN VARCHAR2 DEFAULT 'Basic',
   for_proxy IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 96–34 SET_AUTHENTICATION Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
username (IN)	The username for the HTTP authentication
password (IN)	The password for the HTTP authentication
scheme (IN)	The HTTP authentication scheme. The default, BASIC, denotes the HTTP Basic Authentication scheme.
for_proxy (IN)	Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is FALSE.

Usage Notes

Only the HTTP Basic Authentication scheme is supported.

SET_COOKIE_SUPPPORT Procedure

This procedure enables or disables support for the HTTP cookies in the request. If cookie support is enabled for an HTTP request, all cookies saved in the current session and applicable to the request are returned to the Web server in the request in accordance with HTTP cookie specification standards. Cookies set in the response to the request are saved in the current session for return to the Web server in the subsequent requests if cookie support is enabled for those requests. If the cookie support is disabled for an HTTP request, no cookies are returned to the Web server in the request and the cookies set in the response to the request are not saved in the

current session, although the Set-Cookie HTTP headers can still be retrieved from the response.

Use this procedure to change the cookie support setting a request inherits from the session default setting.

Syntax

```
UTL_HTTP.set_cookie_support(
   r       IN OUT NOCOPY req,
   enable  IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 96–35 SET_COOKIE_SUPPORT Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
enable (IN)	Set enable to TRUE to enable HTTP cookie support; FALSE to disable

Usage Notes

HTTP cookies saved in the current session will last only for the duration of the database session; there is no persistent storage for the cookies. See "UTL_HTTP Examples" on page 96-12 for how to use get_cookies and add_cookies to retrieve, save, and restore cookies.

SET_FOLLOW_REDIRECT Procedure

This procedure sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE function.

If max_redirects is set to a positive number, GET_RESPONSE will automatically follow the redirected URL for the HTTP response status code 301, 302, and 307 for the HTTP HEAD and GET methods, and 303 for all HTTP methods, and retry the HTTP request (the request method will be changed to HTTP GET for the status code 303) at the new location. It follows the redirection until the final, non-redirect location is reached, or an error occurs, or the maximum number of redirections has been reached (to prevent an infinite loop). The url and method fields in the REQ record are updated to the last redirected URL and the method used to access the URL. Set the maximum number of redirects to zero to disable automatic redirection.

Use this procedure to change the maximum number of redirections a request inherits from the session default setting.

Syntax

```
UTL_HTTP.set_follow_redirect(
   r             IN OUT NOCOPY req,
   max_redirects  IN PLS_INTEGER DEFAULT 3);
```

Parameters

Table 96–36 SET_FOLLOW_REDIRECT Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
max_redirects (IN)	The maximum number of redirects. Set to zero to disable redirects.

Usage Notes

The SET_FOLLOW_REDIRECT procedure must be called before GET_RESPONSE for any redirection to take effect.

SET_BODY_CHARSET Procedure

This procedure sets the character set of the request body when the media type is text but the character set is not specified in the Content-Type header. According to the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the "Content-Type" header, the character set of the request or response body should default to "ISO-8859-1".

Use this procedure to change the default body character set a request inherits from the session default setting.

Syntax

```
UTL_HTTP.set_body_charset(
   r        IN OUT NOCOPY req,
   charset  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 96–37 SET_BODY_CHARSET Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
charset (IN)	The default character set of the request body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If charset is NULL, the database character set is assumed.

SET_PERSISTENT_CONN_SUPPORT Procedure

This procedure enables or disables support for the HTTP 1.1 persistent-connection in the request.

If the persistent-connection support is enabled for an HTTP request, the package will keep the network connections to a Web server or the proxy server open in the package after the request is completed properly for a subsequent request to the same server to reuse for each HTTP 1.1 protocol specification. With the persistent connection support, subsequent HTTP requests may be completed faster because the network connection latency is avoided. If the persistent-connection support is disabled for a request, the package will always send the HTTP header "Connection: close" automatically in the HTTP request and close the network connection when the request is completed. This setting has no effect on HTTP requests that follows HTTP 1.0 protocol, for which the network connections will always be closed after the requests are completed.

When a request is being made, the package attempts to reuse an existing persistent connection to the target Web server (or proxy server) if one is available. If none is available, a new network connection will be initiated. The persistent-connection support setting for a request affects only whether the network connection should be closed after a request completes.

Use this procedure to change the persistent-connection support setting a request inherits from the session default setting.

Users should note that while the use of persistent connections in UTL_HTTP may reduce the time it takes to fetch multiple Web pages from the same server, it consumes precious system resources (network connections) in the database server. Also, excessive use of persistent connections may reduce the scalability of the database server when too many network connections are kept open in the database server. Network connections should be kept open only if they will be used

immediately by subsequent requests and should be closed immediately when they are no longer needed. Set the default persistent connection support as disabled in the session, and enable persistent connection in individual HTTP requests as shown in "Example: Using SET_PERSISTENT_CONN_SUPPORT in HTTP Requests" on page 96-46.

Syntax

```
UTL_HTTP.set_persistent_conn_support(
   r       IN OUT NOCOPY req,
   enable  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 96–38 SET_PERSISTENT_CONN_SUPPORT Procedure Parameters

Parameter	Description
`r (IN/OUT)`	The HTTP request
`enable (IN)`	TRUE to keep the network connection persistent. FALSE otherwise.

Usage Notes

The default value of the maximum number of persistent connections in a database session is zero. To truly enable persistent connections, you must also set the maximum number of persistent connections to a positive value or no connections will be kept persistent.

Example: Using SET_PERSISTENT_CONN_SUPPORT in HTTP Requests

```
DECLARE
  TYPE vc2_table IS TABLE OF VARCHAR2(256) INDEX BY binary_integer;
  paths vc2_table;

UTL_HTTP.fetch_pages(paths IN vc2_table) AS
    url_prefix VARCHAR2(256) := 'http://www.my-company.com/';
    req   utl_http.req;
    resp  utl_http.resp;
    data  VARCHAR2(1024);
  BEGIN
    FOR i IN 1..paths.count LOOP
      req := utl_http.begin_request(url_prefix || paths(i));
```

```
      -- Use persistent connection except for the last request
      IF (i < paths.count) THEN
        utl_http.set_persistent_conn_support(req, TRUE);
      END IF;

      resp := utl_http.get_response(req);

      BEGIN
        LOOP
          utl_http.read_text(resp, data);
          -- do something with the data
        END LOOP;
      EXCEPTION
        WHEN utl_http.end_of_body THEN
          NULL;
      END;
      utl_http.end_response(resp);
    END LOOP;
  END;

BEGIN
  utl_http.set_persistent_conn_support(FALSE, 1);
  paths(1) := '...';
  paths(2) := '...';
  ...
  fetch_pages(paths);
END;
```

WRITE_TEXT Procedure

This procedure writes some text data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

Syntax

```
UTL_HTTP.write_text(
   r     IN OUT NOCOPY req,
   data  IN VARCHAR2);
```

Parameters

Table 96–39 WRITE_TEXT Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
data (IN)	The text data to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding: chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the set_header procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the write_raw procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

WRITE_LINE Procedure

This procedure writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP). As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

Syntax

```
UTL_HTTP.write_line(
   r     IN OUT NOCOPY req,
   data  IN VARCHAR2);
```

Parameters

Table 96–40 WRITE_LINE Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
data (IN)	The text line to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. The UTL_HTTP package performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding: chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `set_header` procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the `write_raw` procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

WRITE_RAW Procedure

This procedure writes some binary data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed.

Syntax

```
UTL_HTTP.write_raw(
   r     IN OUT NOCOPY req,
   data  IN RAW);
```

Parameters

Table 96–41 WRITE_RAW Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request
data (IN)	The binary data to send in the HTTP request body

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding: chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `set_header procedure` for details.

END_REQUEST Procedure

This procedure ends the HTTP request. To terminate the HTTP request without completing the request and waiting for the response, the program can call this procedure. Otherwise, the program should go through the normal sequence of beginning a request, getting the response, and closing the response. The network connection will always be closed and will not be reused.

Syntax

```
UTL_HTTP.end_request (
   r  IN OUT NOCOPY req);
```

Parameters

Table 96–42 END_REQUEST Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP request

HTTP Responses

The following APIs manipulate an HTTP response obtained from GET_RESPONSE and receive response information from the Web server. When a response is created for a request, it inherits settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout from the request. Only the body character set can be changed by calling the response API.

GET_RESPONSE Function

This function reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed. The status code, reason phrase, and the HTTP protocol version are stored in the response record. This function completes the HTTP headers section.

Syntax

```
UTL_HTTP.get_response (
   r  IN OUT NOCOPY req)
RETURN resp;
```

Parameters

Table 96–43 GET_RESPONSE Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response

GET_HEADER_COUNT Function

This function returns the number of HTTP response headers returned in the response.

Syntax

```
UTL_HTTP.get_header_count (
   r  IN OUT NOCOPY resp)
RETURN PLS_INTEGER;
```

Parameters

Table 96–44 GET_HEADER_COUNT Function Parameters

Parameter	Description
r (IN/OUT)	The HTTP response

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_HEADER Procedure

This procedure returns the n^{th} HTTP response header name and value returned in the response.

Syntax

```
UTL_HTTP.get_header (
   r      IN OUT NOCOPY resp,
   n      IN PLS_INTEGER,
   name   OUT NOCOPY VARCHAR2,
   value  OUT NOCOPY VARCHAR2);
```

Parameters

Table 96–45 GET_HEADER Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response.
n (IN)	The n^{th} header to return.
name (OUT)	The name of the HTTP response header.
value (OUT)	The value of the HTTP response header.

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_HEADER_BY_NAME Procedure

This procedure returns the HTTP response header value returned in the response given the name of the header.

Syntax

```
UTL_HTTP.get_header_by_name(
   r      IN OUT NOCOPY resp,
   name   IN VARCHAR2,
   value  OUT NOCOPY VARCHAR2,
   n      IN PLS_INTEGER DEFAULT 1);
```

Parameters

Table 96–46 GET_HEADER_BY_NAME Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response
n (IN)	The n^{th} occurrence of an HTTP response header by the specified name to return. The default is 1.

Table 96–46 GET_HEADER_BY_NAME Procedure Parameters

Parameter	Description
`name (IN)`	The name of the HTTP response header for which the value is to return
`value (OUT)`	The value of the HTTP response header.

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

GET_AUTHENTICATION Procedure

This procedure retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.

Syntax

```
UTL_HTTP.get_authentication(
   r          IN OUT NOCOPY resp,
   scheme     OUT VARCHAR2,
   realm      OUT VARCHAR2,
   for_proxy  IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 96–47 GET_AUTHENTICATION Procedure Parameters

Parameter	Description
`r (IN/OUT)`	The HTTP response.
`scheme (OUT)`	The scheme for the required HTTP authentication
`realm (OUT)`	The realm for the required HTTP authentication
`for_proxy (IN)`	Returns the HTTP authentication information required for the access to the HTTP proxy server instead of the Web server? Default is FALSE.

Usage Notes

When a Web client is unaware that a document is protected, at least two HTTP requests are required for the document to be retrieved. In the first HTTP request, the Web client makes the request without supplying required authentication information; so the request is denied. The Web client can determine the authentication information required for the request to be authorized by calling `get_authentication`. The Web client makes the second request and supplies the required authentication information with `set_authorization`. If the authentication information can be verified by the Web server, the request will succeed and the requested document is returned. Before making the request, if the Web client knows that authentication information is required, it can supply the required authentication information in the first request, thus saving an extra request.

SET_BODY_CHARSET Procedure

This procedure sets the character set of the response body when the media type is "text" but the character set is not specified in the "Content-Type" header. For each the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the "Content-Type" header, the character set of the request or response body should default to "ISO-8859-1".

Use this procedure to change the default body character set a response inherits from the request.

Syntax

```
UTL_HTTP.set_body_charset(
   r        IN OUT NOCOPY resp,
   charset  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 96–48 SET_BODY_CHARSET Procedure Parameters

Parameter	Description
`r (IN/OUT)`	The HTTP response.
`charset (IN)`	The default character set of the response body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If `charset` is NULL, the database character set is assumed.

READ_TEXT Procedure

This procedure reads the HTTP response body in text form and returns the output in the caller-supplied buffer. The end_of_body exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

Syntax

```
UTL_HTTP.read_text(
   r     IN OUT NOCOPY resp,
   data  OUT NOCOPY VARCHAR2,
   len   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 96–49 READ_TEXT Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response.
data (OUT)	The HTTP response body in text form
len (IN)	The maximum number of characters of data to read. If len is NULL, this procedure will read as much input as possible to fill the buffer allocated in data. The actual amount of data returned may be less than that specified if little data is available before the end of the HTTP response body is reached or the transfer_timeout amount of time has elapsed. The default is NULL.

Usage Notes

The UTL_HTTP package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, read_text waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, read_text stops reading and returns all the complete multibyte characters read successfully. If

no complete character is read successfully, the partial_multibyte_char exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the read_raw procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the transfer_timeout exception is raised instead. The exception can be handled and the read operation can be retried later.

When the "Content-Type" response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the READ_RAW procedure, or set the character set of the response body explicitly using the SET_BODY_CHARSET procedure and read the response body as text again.

READ_LINE Procedure

This procedure reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer. The end of line is as defined in the function read_line of UTL_TCP. The end_of_body exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

Syntax

```
UTL_HTTP.read_line(
   r            IN OUT NOCOPY resp,
   data         OUT NOCOPY  VARCHAR2,
   remove_crlf  IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 96–50 READ_LINE Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response.
data (OUT)	The HTTP response body in text form
remove_crlf (IN)	Removes the newline characters if set to TRUE

Usage Notes

The UTL_HTTP package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, read_line waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, read_line stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the partial_multibyte_char exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the read_raw procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the transfer_timeout exception is raised instead. The exception can be handled and the read operation can be retried later.

When the "Content-Type" response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the READ_RAW procedure, or set the character set of the response body explicitly using the SET_BODY_CHARSET procedure and read the response body as text again.

READ_RAW Procedure

This procedure reads the HTTP response body in binary form and returns the output in the caller-supplied buffer. The end_of_body exception will be raised if the end of the HTTP response body is reached.

Syntax

```
UTL_HTTP.read_raw(
   r     IN OUT NOCOPY resp,
   data  OUT NOCOPY RAW,
   len   IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 96–51 READ_RAW Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response.
data (OUT)	The HTTP response body in binary form
len (IN)	The number of bytes of data to read. If len is NULL, this procedure will read as much input as possible to fill the buffer allocated in data. The actual amount of data returned may be less than that specified if not much data is available before the end of the HTTP response body is reached or the transfer_timeout amount of time has elapsed. The default is NULL

Usage Notes

The UTL_HTTP package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, read_raw waits for each data packet to be ready to read until timeout occurs. If it occurs, read_raw stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

END_RESPONSE Procedure

This procedure ends the HTTP response. It completes the HTTP request and response. Unless HTTP 1.1 persistent connection is used in this request, the network connection is also closed.

Syntax

```
UTL_HTTP.end_response (
   r  IN OUT NOCOPY resp);
```

Parameters

Table 96–52 END_RESPONSE Procedure Parameters

Parameter	Description
r (IN/OUT)	The HTTP response.

HTTP Cookies

Use the following APIs to manipulate HTTP cookies.

GET_COOKIE_COUNT Function

This function returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers.

Syntax

```
UTL_HTTP.get_cookie_count
RETURN PLS_INTEGER;
```

GET_COOKIES Function

This function returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers.

Syntax

```
UTL_HTTP.get_cookies (
   cookies  IN OUT NOCOPY cookie_table);
```

Parameters

Table 96–53 GET_COOKIES Procedure Parameters

Parameter	Description
cookies (IN/OUT)	The cookies returned

ADD_COOKIES Procedure

This procedure adds the cookies maintained by UTL_HTTP.

Syntax

```
UTL_HTTP.add_cookies (
   cookies  IN cookie_table);
```

Parameters

Table 96–54 ADD_COOKIES Procedure Parameters

Parameter	Description
cookies (IN/OUT)	The cookies to be added

Usage Notes

The cookies that the package currently maintains are not cleared before new cookies are added.

CLEAR_COOKIES Procedure

This procedure clears all cookies maintained by the UTL_HTTP package.

Syntax

```
UTL_HTTP.clear_cookies;
```

HTTP Persistent Connections

Use the following functions to manipulate persistent connections.

GET_PERSISTENT_CONN_COUNT Function

This function returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers.

Syntax

```
UTL_HTTP.get_persistent_conn_count
RETURN PLS_integer;
```

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with

domain names will be counted differently from the host names without domain names.

GET_PERSISTENT_CONNS Procedure

This procedure returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers.

Syntax

```
UTL_HTTP.get_persistent_conns (
   connections  IN OUT NOCOPY connection_table);
```

Parameters

Table 96–55 GET_PERSISTENT_CONNS Procedure Parameters

Parameter	Description
`connections (IN/OUT)`	The network connections kept persistent

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

CLOSE_PERSISTENT_CONN Procedure

This procedure closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session.

Syntax

```
UTL_HTTP.close_persistent_conn (
   conn  IN connection);
```

Parameters

Table 96–56 CLOSE_PERSISTENT_CONN Procedure Parameters

Parameter	Description
conn (IN)	The HTTP persistent connection to close

CLOSE_PERSISTENT_CONNS Procedure

This procedure closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session. This procedure uses a pattern-match approach to decide which persistent connections to close.

To close a group of HTTP persistent connection that share a common property (for example, all connections to a particular host, or all SSL connections), set the particular parameters and leave the rest of the parameters NULL. If a particular parameter is set to NULL when this procedure is called, that parameter will not be used to decide which connections to close.

For example, the following call to the procedure closes all persistent connections to foobar:

```
utl_http.close_persistent_conns(host => 'foobar');
```

And the following call to the procedure closes all persistent connections through the proxy www-proxy at TCP/IP port 80:

```
utl_http.close_persistent_conns(proxy_host => 'foobar',
                                proxy_port => 80);
```

And the following call to the procedure closes all persistent connections:

```
utl_http.close_persistent_conns;
```

Syntax

```
UTL_HTTP.close_persistent_conns (
   host        IN VARCHAR2 DEFAULT NULL,
   port        IN PLS_INTEGER DEFAULT NULL,
   proxy_host  IN VARCHAR2 DEFAULT NULL,
   proxy_port  IN PLS_INTEGER DEFAULT NULL,
   ssl         IN BOOLEAN DEFAULT NULL);
```

Parameters

Table 96–57 CLOSE_PERSISTENT_CONNS Procedure Parameters

Parameter	Description
`host (IN)`	The host for which persistent connections are to be closed
`port (IN)`	The port number for which persistent connections are to be closed
`proxy_host (IN)`	The proxy host for which persistent connections are to be closed
`proxy_port (IN)`	The proxy port for which persistent connections are to be closed
`ssl (IN)`	Close persistent SSL connection

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

Note that the use of a `NULL` value in a parameter when this procedure is called means that the caller does not care about its value when the package decides which persistent connection to close. If you want a `NULL` value in a parameter to match only a `NULL` value of the parameter of a persistent connection (which is when you want to close a specific persistent connection), you should use the `close_persistent_conn` procedure that closes a specific persistent connection.

Error Conditions

The following APIs retrieve error information.

GET_DETAILED_SQLCODE Function

This function retrieves the detailed SQLCODE of the last exception raised.

Syntax

```
UTL_HTTP.get_detailed_sqlcode
RETURN PLS_INTEGER;
```

GET_DETAILED_SQLERRM Function

This function retrieves the detailed SQLERRM of the last exception raised.

Syntax

```
UTL_HTTP.get_detailed_sqlerrm
RETURN VARCHAR2;
```

97

UTL_INADDR

UTL_INADDR provides a PL/SQL procedures to support internet addressing. It provides an API to retrieve host names and IP addresses of local and remote hosts.

This chapter discusses the following topics:

Exceptions

Table 97–1 Exception from Internet Address Package

Exception	Description
UNKNOWN_HOST	The host is unknown.

Summary of UTL_INADDR Subprograms

Table 97–2 UTL_INADDR Subprograms

Subprogram	Description
get_host_name Function on page 97-2	Retrieves the name of the local or remote host given its IP address.
get_host_address Function on page 97-3	Retrieves the IP address of the local or remote host given its name.

get_host_name Function

This function retrieves the name of the local or remote host given its IP address.

Syntax

```
UTL_INADDR.GET_HOST_NAME (
   ip  IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 97–3 get_host_name Function Parameters

Parameter	Description
ip	The IP address of the host used to determine its host name. If ip is not NULL, the official name of the host with its domain name is returned. If this is NULL, the name of the local host is returned and the name does not contain the domain to which the local host belongs.

Returns

The name of the local or remote host of the specified IP address.

Exceptions

`unknown_host`. The specified IP address is unknown.

get_host_address Function

This function retrieves the IP address of a host.

Syntax

```
UTL_INADDR.GET_HOST_ADDRESS (
   host  IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 97–4 get_host_address Function Parameters

Parameter	Description
`host (IN)`	The name of the host to retrieve the IP address. If host is `NULL`, this function returns the IP address of the local host.

98

UTL_RAW

The UTL_RAW package provides SQL functions for manipulating RAW datatypes. This package is necessary because normal SQL functions do not operate on RAWs, and PL/SQL does not allow overloading between a RAW and a CHAR datatype. UTL_RAW also includes subprograms that convert various COBOL number formats to, and from, RAWs.

UTL_RAW is not specific to the database environment, and it may actually be used in other environments as it exists here. For this reason, the prefix UTL has been given to the package, instead of DBMS.

This chapter discusses the following topics:

Usage Notes

UTL_RAW allows a RAW "record" to be composed of many elements. By using the RAW datatype, character set conversion will not be performed, keeping the RAW in its original format when being transferred through remote procedure calls.

With the RAW functions, you can manipulate binary data that was previously limited to the hextoraw and rawtohex functions.

Summary of UTL_RAW Subprograms

Table 98–1 UTL_RAW Subprograms

Subprogram	Description
CAST_FROM_BINARY_INTEGER Function on page 98-3	Returns the binary representation of a BINARY_INTEGER (in RAW).
CAST_FROM_NUMBER Function on page 98-4	Returns the binary representation of a NUMBER (in RAW).
CAST_TO_BINARY_INTEGER Function on page 98-5	Casts the binary representation of a BINARY_INTEGER (in RAW) into a BINARY_INTEGER
CAST_TO_NUMBER Function on page 98-5	Casts the binary representation of a NUMBER (in RAW) into a NUMBER. If include_length is TRUE, the first byte of r encodes the number of bytes in r (
CAST_TO_RAW Function on page 98-6	Converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes.
CAST_TO_VARCHAR2 Function on page 98-7	Converts a RAW represented using n data bytes into VARCHAR2 with n data bytes.
CONCAT Function on page 98-8	Concatenates up to 12 RAWs into a single RAW.
LENGTH Function on page 98-9	Returns the length in bytes of a RAW r.
SUBSTR Function on page 98-9	Returns len bytes, starting at pos from RAW r.
TRANSLATE Function on page 98-11	Translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set.
TRANSLITERATE Function on page 98-12	Converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set.

Table 98–1 UTL_RAW Subprograms (Cont.)

Subprogram	Description
OVERLAY Function on page 98-14	Overlays the specified portion of target RAW with overlay RAW, starting from byte position pos of target and proceding for len bytes.
COPIES Function on page 98-16	Returns n copies of r concatenated together.
XRANGE Function on page 98-17	Returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte.
REVERSE Function on page 98-18	Reverses a byte sequence in RAW r from end to end.
COMPARE Function on page 98-19	Compares RAW r1 against RAW r2.
CONVERT Function on page 98-20	Converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.
BIT_AND Function on page 98-21	Performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.
BIT_OR Function on page 98-22	Performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the "or'd" result RAW.
BIT_XOR Function on page 98-23	Performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the "xor'd" result RAW.
BIT_COMPLEMENT Function on page 98-24	Performs bitwise logical "complement" of the values in RAW r and returns the "complement'ed" result RAW.

CAST_FROM_BINARY_INTEGER Function

This function returns the binary representation of a BINARY_INTEGER (in RAW).

Syntax

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
   n           IN BINARY_INTEGER
   endianess   IN PLS_INTEGER DEFAULT BIG_ENDIAN)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–2 CAST_FROM_BINARY_INTEGER Function Parameters

Parameter	Description
n	The BINARY_INTEGER value.
endianess,	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Returns

The binary representation of the BINARY_INTEGER value.

CAST_FROM_NUMBER Function

This function returns the binary representation of a NUMBER (in RAW). If include_length is TRUE, the first byte of the RAW returned encodes the number of valid bytes in the number (not including the length byte), and the result is padded to a fixed length of 22 bytes with arbitrary data. If include_length is FALSE, the RAW returned is variable length, with a maximum length of 21 bytes.

Syntax

```
UTL_RAW.CAST_FROM_NUMBER (
   n              IN NUMBER
   include_length  IN BOOLEAN)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–3 CAST_FROM_NUMBER Function Parameters

Parameter	Description
n	The NUMBER value.

Returns

The binary representation of the NUMBER value.

CAST_TO_BINARY_INTEGER Function

This function casts the binary representation of a BINARY_INTEGER (in RAW) into a BINARY_INTEGER.

Syntax

```
UTL_RAW.CAST_TO_BINARY_INTEGER (
   r          IN RAW
   endianess  IN PLS_INTEGER DEFAULT BIG_ENDIAN)
RETURN BINARY_INTEGER;
```

Pragmas

```
pragma restrict_references(cast_to_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–4 CAST_TO_BINARY_INTEGER Function Parameters

Parameter	Description
r	The binary representation of a BINARY_INTEGER.
endianess	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

Returns

The BINARY_INTEGER value

CAST_TO_NUMBER Function

This function casts the binary representation of a NUMBER (in RAW) into a NUMBER. If `include_length` is TRUE, the first byte of r encodes the number of bytes in r (not including the length byte) which are valid, up to a maximum of 21 bytes plus the length byte.

Syntax

```
UTL_RAW.CAST_TO_NUMBER (
   r              IN RAW
   include_length  IN BOOLEAN)
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(cast_to_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–5 CAST_TO_NUMBER function Parameters

Parameter	Description
r	The binary representation of a NUMBER

Returns

The NUMBER value.

CAST_TO_RAW Function

This function converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes. The data is not modified in any way; only its datatype is recast to a RAW datatype.

Syntax

```
UTL_RAW.CAST_TO_RAW (
   c  IN VARCHAR2)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–6 CAST_TO_RAW Function Parameters

Parameter	Description
c	VARCHAR2 to be changed to a RAW.

Returns

Table 98–7 CAST_TO_RAW Function Returns

Return	Description
RAW	Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field.
NULL	If c input parameter was NULL.

CAST_TO_VARCHAR2 Function

This function converts a RAW represented using n data bytes into VARCHAR2 with n data bytes.

> **Note:** When casting to a VARCHAR2, the current Globalization Support character set is used for the characters within that VARCHAR2.

Syntax

```
UTL_RAW.CAST_TO_VARCHAR2 (
   r IN RAW)
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(cast_to_varchar2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–8 CAST_TO_VARCHAR2 Function Parameters

Parameter	Description
r	RAW (without leading length field) to be changed to a VARCHAR2).

Returns

Table 98–9 CAST_TO_VARCHAR2 Function Returns

Return	Description
VARCHAR2	Containing having the same data as the input RAW.

Table 98–9 CAST_TO_VARCHAR2 Function Returns

Return	Description
NULL	If r input parameter was NULL.

CONCAT Function

This function concatenates up to 12 RAWs into a single RAW. If the concatenated size exceeds 32K, then an error is returned

Syntax

```
UTL_RAW.CONCAT (
   r1  IN RAW DEFAULT NULL,
   r2  IN RAW DEFAULT NULL,
   r3  IN RAW DEFAULT NULL,
   r4  IN RAW DEFAULT NULL,
   r5  IN RAW DEFAULT NULL,
   r6  IN RAW DEFAULT NULL,
   r7  IN RAW DEFAULT NULL,
   r8  IN RAW DEFAULT NULL,
   r9  IN RAW DEFAULT NULL,
   r10 IN RAW DEFAULT NULL,
   r11 IN RAW DEFAULT NULL,
   r12 IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);
```

Parameters

r1....r12 are the RAW items to concatenate.

Returns

Table 98–10 CONCAT Function Returns

Return	Description
RAW	Containing the items concatenated.

Errors

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a RAW, which is 32767 bytes.

LENGTH Function

This function returns the length in bytes of a RAW r.

Syntax

```
UTL_RAW.LENGTH (
   r  IN RAW)
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–11 LENGTH Function Parameters

Parameter	Description
r	The RAW byte stream to be measured.

Returns

Table 98–12 LENGTH Function Returns

Return	Description
NUMBER	Equal to the current length of the RAW.

SUBSTR Function

This function returns len bytes, starting at pos from RAW r.

Syntax

```
UTL_RAW.SUBSTR (
   r   IN RAW,
   pos IN BINARY_INTEGER,
   len IN BINARY_INTEGER DEFAULT NULL)
```

```
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(substr, WNDS, RNDS, WNPS, RNPS);
```

Parameters

If pos is positive, then SUBSTR counts from the beginning of r to find the first byte. If pos is negative, then SUBSTR counts backward from the end of the r. The value pos cannot be 0.

If len is omitted, then SUBSTR returns all bytes to the end of r. The value len cannot be less than 1.

Table 98–13 SUBSTR Function Parameters

Parameter	Description
r	The RAW byte-string from which a portion is extracted.
pos	The byte position in r at which to begin extraction.
len	The number of bytes from pos to extract from r (optional).

Defaults and Optional Parameters

Table 98–14 SUBSTR Function Exceptions

Optional Parameter	Description
len	Position pos through to the end of r.

Returns

Table 98–15 SUBSTR Function Returns

Return	Description
portion of r	Beginning at pos for len bytes long.
NULL	R input parameter was NULL.

Errors

Table 98–16 SUBSTR Function Errors

Error	Description
VALUE_ERROR	Either pos = 0 or len < 0

TRANSLATE Function

This function translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set. If a byte in r has a matching byte in from_set, then it is replaced by the byte in the corresponding position in to_set, or deleted.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. If to_set is shorter than from_set, then the extra from_set bytes have no translation correspondence and any bytes in r matching.

Note: Difference from TRANSLITERATE:

- Translation RAWs have no defaults.

- r bytes undefined in the to_set translation RAW are deleted.

- Result RAW may be shorter than input RAW r.

Syntax

```
UTL_RAW.TRANSLATE (
   r        IN RAW,
   from_set IN RAW,
   to_set   IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–17 TRANSLATE Function Parameters

Parameter	Description
r	RAW source byte-string to be translated.
from_set	RAW byte-codes to be translated, if present in r.
to_set	RAW byte-codes to which corresponding from_str bytes are translated.

Returns

Table 98–18 TRANSLATE Function Returns

Return	Description
RAW	Translated byte-string.

Errors

Table 98–19 TRANSLATE Function Errors

Error	Description
VALUE_ERROR	Either: - r is NULL or has 0 length - from_set is NULL or has 0 length - to_set is NULL or has 0 length

TRANSLITERATE Function

This function converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set. Successive bytes in r are looked up in the from_set, and, if not found, copied unaltered to the result RAW. If found, then they are replaced in the result RAW by either corresponding bytes in the to_set, or the pad byte when no correspondence exists.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as r.

If the to_set is shorter than the from_set, then the pad byte is placed in the result RAW when a selected from_set byte has no corresponding to_set byte (as if the to_set were extended to the same length as the from_set with pad bytes).

Note: Difference from TRANSLATE:

- r bytes undefined in to_set are padded.

- Result RAW is always same length as input RAW r.

Syntax

```
UTL_RAW.TRANSLITERATE (
   r        IN RAW,
   to_set   IN RAW DEFAULT NULL,
   from_set IN RAW DEFAULT NULL,
   pad      IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–20 TRANSLITERATE Function Parameters

Parameter	Description
r	RAW input byte-string to be converted.
from_set	RAW byte-codes to be converted, if present in r (any length).
to_set	RAW byte-codes to which corresponding from_set bytes are converted (any length).
pad	1 byte used when to-set is shorter than the from_set.

Defaults and Optional Parameters

Table 98–21 TRANSLITERATE Function Optional Parameters

Optional Parameter	Description
from_set	x'00 through x'ff.

Table 98–21 TRANSLITERATE Function Optional Parameters

Optional Parameter	Description
to_set	To the NULL string and effectively extended with pad to the length of from_set as necessary.
pad	x'00'.

Returns

Table 98–22 TRANSLITERATE Function Returns

Return	Description
RAW	Converted byte-string.

Errors

Table 98–23 TRANSLITERATE Function Errors

Error	Description
VALUE_ERROR	R is NULL or has 0 length.

OVERLAY Function

This function overlays the specified portion of target RAW with overlay RAW, starting from byte position pos of target and proceeding for len bytes.

If overlay has less than len bytes, then it is extended to len bytes using the pad byte. If overlay exceeds len bytes, then the extra bytes in overlay are ignored. If len bytes beginning at position pos of target exceeds the length of target, then target is extended to contain the entire length of overlay.

len, if specified, must be greater than, or equal to, 0. pos, if specified, must be greater than, or equal to, 1. If pos exceeds the length of target, then target is padded with pad bytes to position pos, and target is further extended with overlay bytes.

Syntax

```
UTL_RAW.OVERLAY (
   overlay_str IN RAW,
   target      IN RAW,
   pos         IN BINARY_INTEGER DEFAULT 1,
   len         IN BINARY_INTEGER DEFAULT NULL,
```

```
  pad          IN RAW            DEFAULT NULL)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(overlay, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–24 OVERLAY Function Parameters

Parameters	Description
overlay_str	Byte-string used to overlay target.
target	Byte-string which is to be overlayed.
pos	Position in target (numbered from 1) to start overlay.
len	The number of target bytes to overlay.
pad	Pad byte used when overlay len exceeds overlay length or pos exceeds target length.

Defaults and Optional Parameters

Table 98–25 OVERLAY Function Optional Parameters

Optional Parameter	Description
pos	1
len	To the length of overlay
pad	x'00'

Returns

Table 98–26 OVERLAY Function Returns

Return	Description
RAW	The target byte_string overlayed as specified.

Errors

Table 98–27 OVERLAY Function Errors

Error	Description
VALUE_ERROR	Either: - Overlay is NULL or has 0 length - Target is missing or undefined - Length of target exceeds maximum length of a RAW - len < 0 - pos < 1

COPIES Function

This function returns n copies of r concatenated together.

Syntax

```
UTL_RAW.COPIES (
   r IN RAW,
   n IN NUMBER)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–28 COPIES Function Parameters

Parameters	Description
r	RAW to be copied
n	Number of times to copy the RAW (must be positive).

Returns

This returns the RAW copied n times.

Errors

Table 98–29 COPIES Function Errors

Error	Description
VALUE_ERROR	Either: - r is missing, NULL or 0 length - n < 1 - Length of result exceeds maximum length of a RAW

XRANGE Function

This function returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte. If start_byte is greater than end_byte, then the succession of resulting bytes begins with start_byte, wraps through 'FF'x to '00'x, and ends at end_byte. If specified, start_byte and end_byte must be single byte RAWs.

Syntax

```
UTL_RAW.XRANGE (
   start_byte IN RAW DEFAULT NULL,
   end_byte   IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(xrange, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–30 XRANGE Function Parameters

Parameters	Description
start_byte	Beginning byte-code value of resulting sequence.
end_byte	Ending byte-code value of resulting sequence.

Defaults and Optional Parameters

```
start_byte - x'00'
end_byte   - x'FF'
```

Returns

Table 98–31 XRANGE Function Returns

Return	Description
RAW	Containing succession of 1-byte hexadecimal encodings.

REVERSE Function

This function reverses a byte sequence in RAW r from end to end. For example, x'0102F3' would be reversed to x'F30201', and 'xyz' would be reversed to 'zyx'.The result length is the same as the input RAW length.

Syntax

```
UTL_RAW.REVERSE (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(reverse, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–32 REVERSE Function Parameters

Parameter	Description
r	RAW to reverse.

Returns

Table 98–33 REVERSE Function Returns

Return	Description
RAW	Containing the "reverse" of r.

Errors

Table 98–34 REVERSE Function Errors

Error	Description
VALUE_ERROR	R is NULL or has 0 length.

COMPARE Function

This function compares RAW r1 against RAW r2. If r1 and r2 differ in length, then the shorter RAW is extended on the right with pad if necessary.

Syntax

```
UTL_RAW.COMPARE (
   r1  IN RAW,
   r2  IN RAW,
   pad IN RAW DEFAULT NULL)
  RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–35 COMPARE Function Parameters

Parameter	Description
r1	1st RAW to be compared, may be NULL or 0 length.
r2	2nd RAW to be compared, may be NULL or 0 length.
pad	Byte to extend whichever of r1 or r2 is shorter.

Defaults and optional parameters

pad - x'00'

Returns

Table 98–36 COMPARE Function Returns

Return	Description
NUMBER	Equals 0 if RAW byte strings are both NULL or identical; or, Equals position (numbered from 1) of the first mismatched byte.

CONVERT Function

This function converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.

Both from_charset and to_charset must be supported character sets defined to the Oracle server.

Syntax

```
UTL_RAW.CONVERT (
   r            IN RAW,
   to_charset   IN VARCHAR2,
   from_charset IN VARCHAR2)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(convert, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–37 CONVERT Function Parameters

Parameter	Description
r	RAW byte-string to be converted.
to_charset	Name of Globalization Support character set to which r is converted.
from_charset	Name of Globalization Support character set in which r is supplied.

Returns

Table 98–38 CONVERT Function Returns

Return	Description
RAW	Byte string r converted according to the specified character sets.

Errors

Table 98–39 CONVERT Function Errors

Error	Description
VALUE_ERROR	Either: - r missing, NULL, or 0 length - from_charset or to_charset missing, NULL, or 0 length - from_charset or to_charset names invalid or unsupported

BIT_AND Function

This function performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.

If r1 and r2 differ in length, the and operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_AND (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–40 BIT_AND Function Parameters

Parameter	Description
r1	RAW to "and" with r2.
r2	RAW to "and" with r1.

Returns

Table 98–41 BIT_AND Function Returns

Return	Description
RAW	Containing the "and" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

BIT_OR Function

This function performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the or'd result RAW.

If r1 and r2 differ in length, then the "or" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_OR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–42 BIT_OR Function Parameters

Parameters	Description
r1	RAW to "or" with r2.
r2	RAW to "or" with r1.

Returns

Table 98–43 BIT_OR Function Returns

Return	Description
RAW	Containing the "or" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

BIT_XOR Function

This function performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the xor'd result RAW.

If r1 and r2 differ in length, then the "xor" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_XOR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–44 BIT_XOR Function Parameters

Parameter	Description
r1	RAW to "xor" with r2.
r2	RAW to "xor" with r1.

Returns

Table 98–45 BIT_XOR Function Returns

Return	Description
RAW	Containing the "xor" of r1 and r2.
NULL	If either r1 or r2 input parameter was NULL.

BIT_COMPLEMENT Function

This function performs bitwise logical "complement" of the values in RAW r and returns the complement'ed result RAW. The result length equals the input RAW r length.

Syntax

```
UTL_RAW.BIT_COMPLEMENT (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_complement, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 98–46 BIT_COMPLEMENT Function Parameters

Parameter	Description
r	RAW to perform "complement" operation.

Returns

Table 98–47 BIT_COMPLEMENT Function Returns

Return	Description
RAW	The "complement" of r1.
NULL	If r input parameter was NULL.

99

UTL_REF

Oracle8*i* supports user-defined composite type or object type. Any instance of an object type is called an object. An object type can be used as the type of a column or as the type of a table.

In an object table, each row of the table stores an object. You can uniquely identify an object in an object table with an object identifier.

A reference is a persistent pointer to an object, and each reference can contain an object identifier. The reference can be an attribute of an object type, or it can be stored in a column of a table. Given a reference, an object can be retrieved.

The UTL_REF package provides PL/SQL procedures to support reference-based operations. Unlike SQL, UTL_REF procedures enable you to write generic type methods without knowing the object table name.

This chapter discusses the following topics:

Requirements

The procedural option is needed to use this package. This package must be created under SYS (connect / as sysdba). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

Datatypes, Exceptions, and Security for UTL_REF

Datatypes

An object type is a composite datatype defined by the user or supplied as a library type. You can create the object type `employee_type` using the following syntax:

```
CREATE TYPE employee_type AS OBJECT (
   name    VARCHAR2(20),
   id      NUMBER,

member function GET_ID
    (name VARCHAR2)
   RETURN MEMBER);
```

The object type `employee_type` is a user-defined type that contains two attributes, `name` and `id`, and a member function, `GET_ID`().

You can create an object table using the following SQL syntax:

```
CREATE TABLE employee_table OF employee_type;
```

Exceptions

Exceptions can be returned during execution of `UTL_REF` functions for various reasons. For example, the following scenarios would result in exceptions:

- The object selected does not exist. This could be because either:
  1. The object has been deleted, or the given reference is dangling (invalid).
  2. The object table was dropped or does not exist.
- The object cannot be modified or locked in a serializable transaction. The object was modified by another transaction after the serializable transaction started.
- You do not have the privilege to select or modify the object. The caller of the `UTL_REF` subprogram must have the proper privilege on the object that is being selected or modified.

Table 99–1 UTL_REF Exceptions

Exceptions	Description
`errnum == 942`	Insufficient privileges.
`errnum == 1031`	Insufficient privileges.
`errnum == 8177`	Unable to serialize, if in a serializable transaction.
`errnum == 60`	Deadlock detected.
`errnum == 1403`	No data found (if the REF is null, etc.).

The UTL_REF package does not define any named exceptions. You may define exception handling blocks to catch specific exceptions and to handle them appropriately.

Security

You can use the UTL_REF package from stored PL/SQL procedures/packages on the server, as well as from client/side PL/SQL code.

When invoked from PL/SQL procedures/packages on the server, UTL_REF verifies that the invoker has the appropriate privileges to access the object pointed to by the REF.

> **Note:** This is in contrast to PL/SQL packages/procedures on the server which operate with definer's privileges, where the package owner must have the appropriate privileges to perform the desired operations.

Thus, if UTL_REF is defined under user SYS, and user A invokes UTL_REF.SELECT to select an object from a reference, then user A (the invoker) requires the privileges to check.

When invoked from client-side PL/SQL code, UTL_REF operates with the privileges of the client session under which the PL/SQL execution is being done.

Summary of UTL_REF Subprograms

Table 99–2 UTL_REF Subprograms

Subprogram	Description
SELECT_OBJECT Procedure on page 99-4	Selects an object given a reference.
LOCK_OBJECT Procedure on page 99-5	Locks an object given a reference.
UPDATE_OBJECT Procedure on page 99-6	Updates an object given a reference.
DELETE_OBJECT Procedure on page 99-6	Deletes an object given a reference.

SELECT_OBJECT Procedure

This procedure selects an object given its reference. The selected object is retrieved from the database and its value is put into the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
INTO object
FROM object_table t
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.SELECT_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 99–3 SELECT_OBJECT Procedure Parameters

Parameter	Description
reference	Reference to the object to select or retrieve.
object	The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.

Exceptions

May be raised.

LOCK_OBJECT Procedure

This procedure locks an object given a reference. In addition, this procedure lets the program select the locked object. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
  INTO object
  FROM object_table t
  WHERE REF(t) = reference
  FOR UPDATE;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides. It is not necessary to lock an object before updating / deleting it.

Syntax

```
UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>");

UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 99–4 LOCK_OBJECT Procedure Parameters

Parameter	Description
`reference`	Reference of the object to lock.
`object`	The PL / SQL variable that stores the locked object. This variable should be of the same object type as the locked object.

Exceptions

May be raised.

UPDATE_OBJECT Procedure

This procedure updates an object given a reference. The referenced object is updated with the value contained in the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
UPDATE object_table t
SET VALUE(t) = object
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.UPDATE_OBJECT (
   reference IN REF "<typename>",
   object    IN     "<typename>");
```

Parameters

Table 99–5 UPDATE_OBJECT Procedure Parameters

Parameter	Description
`reference`	Reference of the object to update.
`object`	The PL/SQL variable that contains the new value of the object. This variable should be of the same object type as the object to update.

Exceptions

May be raised.

DELETE_OBJECT Procedure

This procedure deletes an object given a reference. The semantic of this subprogram is similar to the following SQL statement:

```
DELETE FROM object_table
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.DELETE_OBJECT (
   reference IN REF "<typename>");
```

Parameters

Table 99–6 DELETE_OBJECT Procedure Parameters

Parameter	Description
reference	Reference of the object to delete.

Exceptions

May be raised.

Example

The following example illustrates usage of the UTL_REF package to implement this scenario: if an employee of a company changes their address, their manager should be notified.

... declarations of Address_t and others...

```
CREATE OR REPLACE TYPE Person_t (
   name    VARCHAR2(64),
   gender  CHAR(1),
   address Address_t,
   MEMBER PROCEDURE setAddress(addr IN Address_t)
);

CREATE OR REPLACE TYPE BODY Person_t (
   MEMBER PROCEDURE setAddress(addr IN Address_t) IS
   BEGIN
      address := addr;
   END;
);

CREATE OR REPLACE TYPE Employee_t (
```

Under Person_t: Simulate implementation of inheritance using a REF to Person_t and delegation of setAddress to it.

```
   thePerson  REF Person_t,
   empno      NUMBER(5),
```

```
   deptREF    Department_t,
   mgrREF     Employee_t,
   reminders  StringArray_t,
   MEMBER PROCEDURE setAddress(addr IN Address_t),
   MEMBER procedure addReminder(reminder VARCHAR2);
);

CREATE TYPE BODY Employee_t (
   MEMBER PROCEDURE setAddress(addr IN Address_t) IS
      myMgr Employee_t;
      meAsPerson Person_t;
   BEGIN
```

Update the address by delegating the responsibility to `thePerson`. Lock the Person object from the reference, and also select it:

```
      UTL_REF.LOCK_OBJECT(thePerson, meAsPerson);
      meAsPerson.setAddress(addr);
```

Delegate to `thePerson`:

```
        UTL_REF.UPDATE_OBJECT(thePerson, meAsPerson);
        if mgr is NOT NULL THEN
```

Give the manager a reminder:

```
         UTL_REF.LOCK_OBJECT(mgr);
         UTL_REF.SELECT_OBJECT(mgr, myMgr);
         myMgr.addReminder
         ('Update address in the employee directory for' ||
         thePerson.name || ', new address: ' || addr.asString);
         UTL_REF.UPDATE_OBJECT(mgr, myMgr);
      END IF;
   EXCEPTION
      WHEN OTHERS THEN
      errnum := SQLCODE;
      errmsg := SUBSTR(SQLERRM, 1, 200);
```

100

UTL_SMTP

UTL_SMTP is designed for sending e-mail over Simple Mail Transfer Protocol (SMTP). It does not have the functionality to implement an SMTP server for mail clients to send e-mail using SMTP.

Many interfaces to the SMTP package appear as both a function and a procedure. The functional form returns the reply from the server for processing by the client. The procedural form discards the reply but raises an exception if the reply indicates a transient (400-range reply code) or permanent error (500-range reply code).

Note that the original SMTP protocol communicates using 7-bit ASCII. Using UTL_SMTP, all text data (in other words, those in VARCHAR2) will be converted to US7ASCII before it is sent over the wire to the server. Some implementations of SMTP servers that support SMTP extension 8BITMIME [RFC1652] support full 8-bit communication between client and server.

The body of the DATA command may be transferred in full 8 bits, but the rest of the SMTP command and response should be in 7 bits. When the target SMTP server supports 8BITMIME extension, users of multibyte databases may convert their non-US7ASCII, multibyte VARCHAR2 data to RAW and use the `write_raw_data()` API to send multibyte data using 8-bit MIME encoding.

UTL_SMTP provides for SMTP communication as specified in RFC821, but does not provide an API to format the content of the message according to RFC 822 (for example, setting the subject of an electronic mail).You must format the message appropriately.

This chapter discusses the following topics:

Note : RFC documents are "Request for Comments" documents that describe proposed standards for public review on the Internet. For the actual RFC documents, please refer to:

`http://www.ietf.org/rfc/`

Exceptions, Limitations, and Reply Codes

Exceptions

Table 100–1 lists the exceptions that can be raised by the API of the `UTL_SMTP` package. The network error is transferred to a reply code of 421- service not available.

Table 100–1 UTL_SMTP Exceptions

Exception	Description
`INVALID_OPERATION`	Raised when an invalid operation is made. In other words, calling API other than `write_data(), write_raw_data()` or `close_data()` after open_data() is called, or calling `write_data(), write_raw_data()` or `close_data()` without first calling `open_data()`.
`TRANSIENT_ERROR`	Raised when receiving a reply code in 400 range.
`PERMANENT_ERROR`	Raised when receiving a reply code in 500 range.

Limitations

No limitation or range-checking is imposed by the API. However, you should be aware of the following size limitations on various elements of SMTP. Sending data that exceed these limits may result in errors returned by the server.

Table 100–2 SMTP Size Limitation

Element	Size Limitation
user	The maximum total length of a user name is 64 characters.
domain	The maximum total length of a domain name or number is 64 characters.
path	The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).

Table 100–2 SMTP Size Limitation

Element	Size Limitation
command line	The maximum total length of a command line including the command word and the <CRLF> is 512 characters.
reply line	The maximum total length of a reply line including the reply code and the <CRLF> is 512 characters.
text line	The maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency).
recipients buffer	The maximum total number of recipients that must be buffered is 100 recipients.

Reply Codes

The following is a list of the SMTP reply codes.

Table 100–3 SMTP Reply Codes

Reply Code	Meaning
211	System status, or system help reply
214	Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220	<domain> Service ready
221	<domain> Service closing transmission channel
250	Requested mail action okay, completed
251	User not local; will forward to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
253	OK, <messages> pending messages for node <node> started
354	Start mail input; end with <CRLF>.<CRLF>
355	Octet-offset is the transaction offset
421	<domain> Service not available, closing transmission channel (This may be a reply to any command if the service knows it must shut down.)

Table 100–3 SMTP Reply Codes

Reply Code	Meaning
450	Requested mail action not taken: mailbox unavailable [for example, mailbox busy]
451	Requested action aborted: local error in processing
452	Requested action not taken: insufficient system storage
453	You have no mail.
454	TLS not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: reason
500	Syntax error, command unrecognized (This may include errors such as command line too long.)
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<Machine> does not accept mail.
530	Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.
534	Authentication mechanism is too weak.
538	Encryption required for requested authentication mechanism.
550	Requested action not taken: mailbox unavailable [for , mailbox not found, no access]
551	User not local; please try <forward-path>
552	Requested mail action aborted: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed [for example, mailbox syntax incorrect]
554	Transaction failed

Summary of UTL_SMTP Subprograms

Table 100–4 UTL_SMTP Subprograms

Subprogram	Description
connection Record Type on page 100-6	This is a PL/SQL record type used to represent a SMTP connection.
reply, replies Record Types on page 100-7	PL/SQL record types used to represent an SMTP reply line.
open_connection Function on page 100-7	Opens a connection to an SMTP server.
command(), command_replies() Functions on page 100-8	Performs a generic SMTP command.
helo Function on page 100-9	Performs initial handshaking with SMTP server after connecting.
ehlo Function on page 100-10	Performs initial handshaking with SMTP server after connecting, with extended information returned.
mail Function on page 100-11	Initiates a mail transaction with the server. The destination is a mailbox.
rcpt Function on page 100-12	Specifies the recipient of an e-mail message.
data Function on page 100-13	Specifies the body of an e-mail message.
open_data(), write_data(), write_raw_data(), close_data() Functions on page 100-14	Provide more fine-grain control to the data() API.
rset Function on page 100-15	Aborts the current mail transaction.
vrfy Function on page 100-16	Verifies the validity of a destination e-mail address.
noop() Function on page 100-17	The null command.
quit Function on page 100-18	Terminates an SMTP session and disconnects from the server.

connection Record Type

This is a PL/SQL record type used to represent an SMTP connection.

Syntax

```
TYPE connection IS RECORD (
    host              VARCHAR2(255),      -- remote host name
    port              PLS_INTEGER,        -- remote port number
    tx_timeout        PLS_INTEGER,        -- Transfer time-out (in seconds)
    private_tcp_con   utl_tcp.connection, -- private, for implementation use
    private_state     PLS_INTEGER         -- private, for implementation use
);
```

Fields

Table 100–5 connection Record Type Fields

Field	Description
`host`	The name of the remote host when connection is established. `NULL` when no connection is established.
`port`	The port number of the remote SMTP server connected. `NULL` when no connection is established.
`tx_timeout`	The time in seconds that the `UTL_SMTP` package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent into the network without being blocked. `0` indicates not to wait at all. `NULL` indicates to wait forever.
`private_tcp_con`	Private, for implementation use only. You should not modify this field.
`private_state`	Private, for implementation use only. You should not modify this field.

Usage Notes

The read-only fields in a connection record are used to return information about the SMTP connection after the connection is successfully made with `open_connection()`. Changing the values of these fields has no effect on the connection. The fields `private_xxx` are for implementation use only. You should not modify these fields.

reply, replies Record Types

These are PL/SQL record types used to represent an SMTP reply line. Each SMTP reply line consists of a reply code followed by a text message. While a single reply line is expected for most SMTP commands, some SMTP commands expect multiple reply lines. For those situations, a PL/SQL table of reply records is used to represent multiple reply lines.

Syntax

```
TYPE reply IS RECORD (
  code    PLS_INTEGER,       -- 3-digit reply code
  text    VARCHAR2(508)      -- text message
);
TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;   -- multiple reply
lines
```

Fields

Table 100–6 reply, replies Record Type Fields

Field	Description
code	The 3-digit reply code.
text	The text message of the reply.

open_connection Function

This function opens a connection to an SMTP server.

Syntax

```
UTL_SMTP.OPEN_CONNECTION (
   host        IN  VARCHAR2,
   port        IN  PLS_INTEGER DEFAULT 25,
   c           OUT connection,
   tx_timeout  IN  PLS_INTEGER DEFAULT NULL)
RETURN reply;
UTL_SMTP.OPEN_+CONNECTION (
   host        IN  VARCHAR2,
   port        IN  PLS_INTEGER DEFAULT 25,
   tx_timeout  IN  PLS_INTEGER DEFAULT NULL)
RETURN connection;
```

Parameters

Table 100–7 open_connection Function Parameters

Parameter	Description
host (IN)	The name of the SMTP server host
port (IN)	The port number on which SMTP server is listening (usually 25).
tx_timeout (IN)	The time in seconds that the UTL_SMTP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.

Usage Notes

The expected response from the server is a message beginning with status code 220.

The version of open_connection() API that returns utl_smtp.connection record is actually the procedure version of open_connection that checks the reply code returned by an SMTP server when the connection is first established.

A timeout on the write operations feature is not supported in the current release of this package.

command(), command_replies() Functions

These functions perform generic SMTP commands.

Syntax

```
UTL_SMTP.COMMAND (
   c    IN connection,
   cmd  IN VARCHAR2,
   arg  IN VARCHAR2 DEFAULT NULL)
RETURN reply;
UTL_SMTP.COMMAND (
   c    IN connection,
   cmd  IN VARCHAR2,
   arg  IN ARCHAR2 DEFAULT NULL);
UTL_SMTP.COMMAND_REPLIES (
   c    IN connection,
```

```
   cmd  IN VARCHAR2,
   arg  IN VARCHAR2 DEFAULT NULL)
RETURN replies;
```

Parameters

Table 100–8 command (), command_replies () Function Parameters

Parameter	Description
c (IN)	The SMTP connection.
cmd (IN)	The SMTP command to send to the server.
arg (IN)	The optional argument to the SMTP argument. A space will be inserted between cmd and arg.

Usage Notes

These are the APIs used to invoke generic SMTP commands. Use command() if only a single reply line is expected. Use command_replies() if multiple reply lines are expected (in other words, EXPN or HELP).

For command(), if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

helo Function

This function performs initial handshaking with SMTP server after connecting.

Syntax

```
UTL_SMTP.HELO (
   c  IN NOCOPY connection, domain IN NOCOPY)
RETURN reply;
UTL_SMTP.HELO (
   c  IN NOCOPY connection, domain IN NOCOPY);
```

Parameters

Table 100–9 helo Function Parameters

Parameter	Description
c (IN NOCOPY)	The SMTP connection.

Table 100–9 helo Function Parameters

Parameter	Description
`domain (IN NOCOPY)`	The domain name of the local (sending) host. Used for identification purposes.

Usage Notes

RFC 821 specifies that the client must identify itself to the server after connecting. This routine performs that identification. The connection must have been opened via a call to `open_connection()` before calling this routine.

The expected response from the server is a message beginning with status code 250.

Related Functions

`ehlo()`

ehlo Function

This function performs initial handshaking with SMTP server after connecting, with extended information returned.

Syntax

```
UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN NOCOPY)
RETURN replies;
UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN NOCOPY);
```

Parameters

Table 100–10 ehlo Function Parameters

Parameter	Description
`c (IN NOCOPY)`	The SMTP connection.
`domain (IN NOCOPY)`	The domain name of the local (sending) host. Used for identification purposes.

Usage Notes

The `ehlo()` interface is identical to `helo()`, except that it allows the server to return more descriptive information about its configuration. [RFC1869] specifies the format of the information returned, which the PL/SQL application can retrieve using the functional form of this call. For compatibility with `helo()`, each line of text returned by the server begins with status code 250.

Related Functions

`helo()`

mail Function

This function initiates a mail transaction with the server. The destination is a mailbox.

Syntax

```
UTL_SMTP.MAIL (
   c           IN OUT NOCOPY connection,
   sender      IN OUT NOCOPY,
   parameters  IN OUT NOCOPY)
RETURN reply;
UTL_SMTP.MAIL (
   c           IN OUT NOCOPY connection,
   sender      IN OUT NOCOPY,
   parameters  IN OUT NOCOPY);
```

Parameters

Table 100–11 Mail Function Parameters

Parameter	Description
`c (IN NOCOPY)`	The SMTP connection.
`sender (IN OUT NOCOPY)`	The e-mail address of the user sending the message.
`parameters (IN OUT NOCOPY)`	The additional parameters to MAIL command as defined in Section 6 of [RFC1869]. It should follow the format of "XXX=XXX (XXX=XXX ....)".

Usage Notes

This command does not send the message; it simply begins its preparation. It must be followed by calls to `rcpt()` and `data()` to complete the transaction. The connection to the SMTP server must be open and a `helo()` or `ehlo()` command must have already been sent.

The expected response from the server is a message beginning with status code 250.

rcpt Function

This function specifies the recipient of an e-mail message.

Syntax

```
UTL_SMTP.RCPT (
   c           IN OUT NOCOPY connection,
   recipient   IN OUT NOCOPY,
   parameters  IN OUT NOCOPY)
RETURN reply;
UTL_SMTP.RCPT (
   c           IN OUT NOCOPY connection
   recipient   IN OUT NOCOPY,
   parameters  IN OUT NOCOPY);
```

Table 100–12 rcpt Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The SMTP connection.
`recipient (IN OUT NOCOPY)`	The e-mail address of the user to which the message is being sent.
`parameters (IN OUT NOCOPY)`	The additional parameters to RCPT command as defined in Section 6 of [RFC1869]. It should follow the format of "XXX=XXX (XXX=XXX ....)".

Usage Notes

To send a message to multiple recipients, call this routine multiple times. Each invocation schedules delivery to a single e-mail address. The message transaction must have been begun by a prior call to `mail()`, and the connection to the mail server must have been opened and initialized by prior calls to `open_connection()` and `helo()` or `ehlo()`, respectively.

The expected response from the server is a message beginning with status code 250 or 251.

data Function

This function specifies the body of an e-mail message.

Syntax

```
UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN OUT NOCOPY)
RETURN reply;
UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN OUT NOCOPY);
```

Parameters

Table 100–13 data Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The SMTP Connection.
`body (IN OUT NOCOPY)`	The text of the message to be sent, including headers, in [RFC822] format.

Usage Notes

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The `data()` routine will terminate the message with a <CR><LF>.<CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It will also translate any sequence of <CR><LF>.<CR><LF> (single period) in body to <CR><LF>..<CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

The `data()` call should be called only after `open_connection()`, `helo() / ehlo()`, `mail()` and `rcpt()` have been called. The connection to the SMTP server must be open, and a mail transaction must be active when this routine is called.

The expected response from the server is a message beginning with status code 250. The 354 response received from the initial DATA command will not be returned to the caller.

open_data(), write_data(), write_raw_data(), close_data() Functions

These APIs provide more fine-grain control to the `data()` API; in other words, to the SMTP DATA operation. `open_data()` sends the DATA command. After that, `write_data()` and `write_raw_data()` write a portion of the e-mail message. A repeat call to `write_data()` and `write_raw_data()` appends data to the e-mail message. The `close_data()` call ends the e-mail message by sending the sequence <CR><LF>.<CR><LF> (a single period at the beginning of a line).

Syntax

```
UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;
UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection);
UTL_SMTP.WRITE_DATA (
   c     IN OUT NOCOPY connection,
   data  IN OUT NOCOPY);
UTL_SMTP.WRITE_RAW_DATA (
   c     IN OUT NOCOPY connection
   data  IN OUT NOCOPY);
UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;
UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection);
```

Parameters

Table 100–14 open_data(), write_data(), write_raw_data(), close_data() Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The SMTP connection.
`data (IN OUT NOCOPY)`	The portion of the text of the message to be sent, including headers, in [RFC822] format.

Usage Notes

The calls to `open_data()`, `write_data()`, `write_raw_data()` and `close_data()` must be made in the right order. A program calls `open_data()` to send the DATA command to the SMTP server. After that, it can call `write_data()` or

write_raw_data() repeatedly to send the actual data. The data is terminated by calling close_data(). After open_data() is called, the only APIs that can be called are write_data(), write_raw_data(), or close_data(). A call to other APIs will result in an INVALID_OPERATION exception being raised.

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The data() routine will terminate the message with a <CR><LF>.<CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It will also translate any sequence of <CR><LF>.<CR><LF> (single period) in the body to <CR><LF>..<CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

Notice that this conversion is not bullet-proof. Consider this code fragment:

```
utl_smtp.write_data('some message.' || chr(13) || chr(10));
utl_smtp.write_data('.' || chr(13) || chr(10));
```

Since the sequence <CR><LF>.<CR><LF> is split between two calls to write_data(), the implementation of write_data() will not detect the presence of the data-terminator sequence, and therefore, will not perform the translation. It will be the responsibility of the user to handle such a situation, or it may result in premature termination of the message data.

XXX_data() should be called only after open_connection(), helo()/ehlo(), mail(), and rcpt() have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of write_data() because the SMTP server does not respond until the data-terminator is sent during the call to close_data().

Text (VARCHAR2) data sent using write_data() API is converted to US7ASCII before it is sent. If the text contains multibyte characters, each multibyte character in the text that cannot be converted to US7ASCII is replaced by a '?' character. If 8BITMIME extension is negotiated with the SMTP server using the EHLO() API, multibyte VARCHAR2 data can be sent by first converting the text to RAW using the UTL_RAW package, and then sending the RAW data using write_raw_data().

rset Function

This function aborts the current mail transaction.

Syntax

```
UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection)
RETURN reply;
UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection);
```

Parameters

Table 100–15 rset Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The SMTP connection.

Usage Notes

This command allows the client to abandon a mail message it was in the process of composing. No mail will be sent. The client can call `rset()` at any time after the connection to the SMTP server has been opened via `open_connection()`. The server will always respond to `RSET` with a message beginning with status code 250.

Related Functions

`quit()`

vrfy Function

This function verifies the validity of a destination e-mail address.

Syntax

```
UTL_SMTP.VRFY (
   c          IN OUT NOCOPY connection
   recipient  IN OUT NOCOPY)
RETURN reply;
```

Parameters

Table 100–16 vrfy Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The SMTP connection.

Table 100–16 vrfy Function Parameters

Parameter	Description
`recipient (IN OUT NOCOPY)`	The e-mail address to be verified.

Usage Notes

The server attempts to resolve the destination address `recipient`. If successful, it returns the recipient's full name and fully qualified mailbox path. The connection to the server must have already been established via `open_connection()` and `helo()` / `ehlo()` before making this request.

Successful verification returns one or more lines beginning with status code 250 or 251.

Related Functions

`expn()`

noop() Function

The null command.

Syntax

```
UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection)
RETURN VARCHAR2;
UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection);
```

Parameter

Table 100–17 noop Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The SMTP connection.

Usage Notes

This command has no effect except to elicit a successful reply from the server. It can be issued at any time after the connection to the server has been established with

open_connection(). The noop() command can be used to verify that the server is still connected and is listening properly.

This command will always reply with a single line beginning with status code 250.

quit Function

This function terminates an SMTP session and disconnects from the server.

Syntax

```
UTL_SMTP.QUIT (
   c  IN OUT NOCOPY connection)
RETURN VARCHAR2;
```

Parameter

Table 100–18 quit Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The SMTP connection.

Usage Notes

The quit() command informs the SMTP server of the client's intent to terminate the session. It then closes the connection established by open_connection(), which must have been called before executing this command. If a mail transaction is in progress when quit() is issued, it is abandoned in the same manner as rset().

The function form of this command returns a single line beginning with the status code 221 on successful termination. In all cases, the connection to the SMTP server is closed. The fields remote_host and remote_port of c are reset.

Related Functions

rset()

Example

The following example illustrates how UTL_SMTP is used by an application to send e-mail. The application connects to an SMTP server at port 25 and sends a simple text message.

```
DECLARE
  c utl_smtp.connection;

  PROCEDURE send_header(name IN VARCHAR2, header IN VARCHAR2) AS
  BEGIN
    utl_smtp.write_data(c, name || ': ' || header || utl_tcp.CRLF);
  END;

BEGIN
  c := utl_smtp.open_connection('smtp-server.acme.com');
  utl_smtp.helo(c, 'foo.com');
  utl_smtp.mail(c, 'sender@foo.com');
  utl_smtp.rcpt(c, 'recipient@foo.com');
  utl_smtp.open_data(c);
  send_header('From',    '"Sender" <sender@foo.com>');
  send_header('To',      '"Recipient" <recipient@foo.com>');
  send_header('Subject', 'Hello');
  utl_smtp.write_data(c, utl_tcp.CRLF || 'Hello, world!');
  utl_smtp.close_data(c);
  utl_smtp.quit(c);
EXCEPTION
  WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN
    BEGIN
      utl_smtp.quit(c);
    EXCEPTION
      WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN
        NULL; -- When the SMTP server is down or unavailable, we don't have
              -- a connection to the server. The quit call will raise an
              -- exception that we can ignore.
    END;
    raise_application_error(-20000,
      'Failed to send mail due to the following error: ' || sqlerrm);
END;
```

101

UTL_TCP

With the UTL_TCP package and its procedures and functions, PL/SQL applications can communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols and e-mail.

The UTL_TCP package provides TCP/IP client-side access functionality in PL/SQL. The API provided in the package only allows connections to be initiated by the PL/SQL program. It does not allow the PL/SQL program to accept connections initiated outside the program.

This chapter discusses the following topics:

Exceptions

The exceptions raised by the TCP/IP package are listed in Table 101–1.

Table 101–1 TCP/IP Exceptions

Exception	Description
BUFFER_TOO_SMALL	Buffer is too small for input that requires look-ahead.
END_OF_INPUT	Raised when no more data is available to read from the connection.
NETWORK_ERROR	Generic network error.
BAD_ARGUMENT	Bad argument passed in an API call (for example, a negative buffer size).
TRANSFER_TIMEOUT	No data is read and a read time-out occurred.
PARTIAL_MULTIBYTE_CHAR	No complete character is read and a partial multibyte character is found at the end of the input.

Example

The following code example illustrates how the TCP/IP package can be used to retrieve a Web page over HTTP. It connects to a Web server listening at port 80 (standard port for HTTP) and requests the root document.

```
DECLARE
  c  utl_tcp.connection;  -- TCP/IP connection to the Web server
  ret_val pls_integer;
BEGIN
  c := utl_tcp.open_connection(remote_host => 'www.acme.com',
                               remote_port =>  80,
                               charset     => 'US7ASCII');  -- open connection
  ret_val := utl_tcp.write_line(c, 'GET / HTTP/1.0');    -- send HTTP request
  ret_val := utl_tcp.write_line(c);
  BEGIN
    LOOP
      dbms_output.put_line(utl_tcp.get_line(c, TRUE));  -- read result
    END LOOP;
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      NULL; -- end of input
  END;
  utl_tcp.close_connection(c);
END;
```

The following code example illustrates how the TCP/IP package can be used by an application to send e-mail (also known as email from PL/SQL). The application connects to an SMTP server at port 25 and sends a simple text message.

```
PROCEDURE send_mail (sender    IN VARCHAR2,
                     recipient IN VARCHAR2,
                    message   IN VARCHAR2)
IS
    mailhost   VARCHAR2(30) := 'mailhost.mydomain.com';
    smtp_error  EXCEPTION;
    mail_conn   utl_tcp.connection;
    PROCEDURE smtp_command(command IN VARCHAR2,
                           ok      IN VARCHAR2 DEFAULT '250')
    IS
        response varchar2(3);
        len pls_integer;
    BEGIN
        len := utl_tcp.write_line(mail_conn, command);
        response := substr(utl_tcp.get_line(mail_conn), 1, 3);
        IF (response <> ok) THEN
            RAISE smtp_error;
        END IF;
    END;

BEGIN
    mail_conn := utl_tcp.open_connection(remote_host => mailhost,
                                         remote_port => 25,
                                         charset     => 'US7ASCII');
    smtp_command('HELO ' || mailhost);
    smtp_command('MAIL FROM: ' || sender);
    smtp_command('RCPT TO: ' || recipient);
    smtp_command('DATA', '354');
    smtp_command(message);
    smtp_command('QUIT', '221');
    utl_tcp.close_connection(mail_conn);
EXCEPTION
    WHEN OTHERS THEN
        -- Handle the error
END;
```

Summary of UTL_TCP Subprograms

Table 101–2 UTL_TCP Subprograms

Subprogram	Description
connection on page 101-4	A PL/SQL record type used to represent a TCP/IP connection.
CRLF on page 101-6	The character sequence carriage-return line-feed. It is the newline sequence commonly used many communication standards.
open_connection Function on page 101-6	Opens a TCP/IP connection to a specified service.
available Function on page 101-9	Determines the number of bytes available for reading from a TCP/IP connection.
read_raw Function on page 101-10	Receives binary data from a service on an open connection.
write_raw Function on page 101-11	Transmits a binary message to a service on an open connection.
read_text Function on page 101-12	Receives text data from a service on an open connection.
write_text Function on page 101-14	Transmits a text message to a service on an open connection.
read_line Function on page 101-15	Receives a text line from a service on an open connection.
write_line Function on page 101-16	Transmits a text line to a service on an open connection.
get_raw(), get_text(), get_line() Functions on page 101-17	Convenient forms of the read functions, which return the data read instead of the amount of data read.
flush Procedure on page 101-18	Transmits all data in the output buffer, if a buffer is used, to the server immediately.
close_connection Procedure on page 101-18	Closes an open TCP/IP connection.
close_all_connections Procedure on page 101-19	Closes all open TCP/IP connections.

connection

This is a PL/SQL record type used to represent a TCP/IP connection.

Syntax

```
TYPE connection IS RECORD (
    remote_host    VARCHAR2(255), -- remote host name
    remote_port    PLS_INTEGER,   -- remote port number
    local_host     VARCHAR2(255), -- local host name
    local_port     PLS_INTEGER,   -- local port number
    charset        VARCHAR2(30),  -- character set for on-the-wire communication
    newline        VARCHAR2(2),   -- newline character sequence
    tx_timeout     PLS_INTEGER,   -- transfer time-out value (in seconds)
    private_sd     PLS_INTEGER,   -- for internal use
    );
```

Fields

Table 101–3 connection Record Type Fields

Field	Description
`remote_host`	The name of the remote host when connection is established. NULL when no connection is established.
`remote_port`	The port number of the remote host connected. NULL when no connection is established.
`local_host`	The name of the local host used to establish the connection. NULL when no connection is established.
`local_port`	The port number of the local host used to establish the connection. NULL when no connection is established.
`charset`	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database will be converted to and from the on-the-wire character set as they are sent and received on the network.
`newline`	The newline character sequence. This newline character sequence is appended to the text line sent by write_line() API.
`tx_timeout`	A time in seconds that the UTL_TCP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Usage Notes

The fields in a connection record are used to return information about the connection, which is often made using `open_connection()`. Changing the values of those fields has no effect on the connection. The fields `private_XXXX` are for implementation use only. You should not modify the values.

In the current release of the `UTL_TCP` package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields will not be set in the connection record returned by the function.

Time-out on write operations is not supported in the current release of the `UTL_TCP` package.

CRLF

The character sequence carriage-return line-feed. It is the newline sequence commonly used many communication standards.

Syntax

```
CRLF varchar2(10);
```

Usage Notes

This package variable defines the newline character sequence commonly used in many Internet protocols. This is the default value of the newline character sequence for `write_line()`, specified when a connection is opened. While such protocols use `<CR><LF>` to denote a new line, some implementations may choose to use just line-feed to denote a new line. In such cases, users can specify a different newline character sequence when a connection is opened.

This CRLF package variable is intended to be a constant that denotes the carriage-return line-feed character sequence. Do not modify its value. Modification may result in errors in other PL/SQL applications.

open_connection Function

This function opens a TCP/IP connection to a specified service.

Syntax

```
UTL_TCP.OPEN_CONNECTION  (remote_host      IN VARCHAR2,
```

```
remote_port      IN PLS_INTEGER,
local_host       IN VARCHAR2 DEFAULT NULL,
local_port       IN PLS_INTEGER DEFAULT NULL,
in_buffer_size   IN PLS_INTEGER DEFAULT NULL,
out_buffer_size  IN PLS_INTEGER DEFAULT NULL,
charset          IN VARCHAR2 DEFAULT NULL,
newline          IN VARCHAR2 DEFAULT CRLF,
tx_timeout       IN PLS_INTEGER DEFAULT NULL)
RETURN connection;
```

Parameters

Table 101–4 open_connection Function Parameters

Parameter	Description
`remote_host (IN)`	The name of the host providing the service. When `remote_host` is NULL, it connects to the local host.
`remote_port (IN)`	The port number on which the service is listening for connections.
`local_host (IN)`	The name of the host providing the service. NULL means don't care.
`local_port (IN)`	The port number on which the service is listening for connections. NULL means don't care.
`in_buffer_size (IN)`	The size of input buffer. The use of an input buffer can speed up execution performance in receiving data from the server. The appropriate size of the buffer depends on the flow of data between the client and the server, and the network condition. A 0 value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the input buffer is 32767 bytes.
`out_buffer_size (IN)`	The size of output buffer. The use of an output buffer can speed up execution performance in sending data to the server. The appropriate size of buffer depends on the flow of data between the client and the server, and the network condition. A 0 value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the output buffer is 32767 bytes.

Table 101–4 open_connection Function Parameters

Parameter	Description
`charset (IN)`	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database will be converted to and from the on-the-wire character set as they are sent and received on the network using `read_text(), read_line(), write_text()` and `write_line()`. Set this parameter to NULL when no conversion is needed.
`newline (IN)`	The newline character sequence. This newline character sequence is appended to the text line sent by `write_line()` API.
`tx_timeout`	A time in seconds that the `UTL_TCP` package should wait before giving up in a read or write operations in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Usage Notes

Note that connections opened by this UTL_TCP package can remain open and be passed from one database call to another in a shared server configuration. However, the connection must be closed explicitly. The connection will remain open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

The parameters local_host and local_port are ignored currently when open_connection makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made.

In the current release of the `UTL_TCP` package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields will not be set in the connection record returned by the function.

Time-out on write operations is not supported in the current release of the UTL_TCP package.

Related Functions

close_connection(), close_all_connections()

available Function

This function determines the number of bytes available for reading from a TCP/IP connection. It is the number of bytes that can be read immediately without blocking. Determines if data is ready to be read from the connection.

Syntax

```
UTL_TCP.AVAILABLE (
   c        IN OUT NOCOPY connection,
   timeout  IN PLS_INTEGER DEFAULT 0)
RETURN PLS_INTEGER;
```

Parameters

Table 101–5 Available Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The TCP connection to determine the amount of data that is available to be read from.
timeout	A time in seconds to wait before giving up and reporting that no data is available. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

Usage Notes

The connection must have already been opened through a call to open_connection(). Users may use this API to determine if data is available to be read before calling the read API so that the program will not be blocked because data is not ready to be read from the input.

The number of bytes available for reading returned by this function may less than than what is actually available. On some platforms, this function may only return 1, to indicate that some data is available. If you are concerned about the portability of your application, assume that this function returns a positive value when data is available for reading, and 0 when no data is available. The following example illustrates using this function in a portable manner:

```
DECLARE
   c   utl_tcp.connection
```

```
   data VARCHAR2(256);
   len  PLS_INTEGER;
BEGIN
   c := utl_tcp.open_connection(...);
   LOOP
      IF (utl_tcp.available(c) > 0) THEN
         len := utl_tcp.read_text(c, data, 256);
      ELSE
         ---do some other things
        . . . .
      END IF
   END LOOP;
END;
```

Related Functions

`read_raw()`, `read_text()`, `read_line()`

read_raw Function

This function receives binary data from a service on an open connection.

Syntax

```
UTL_TCP.READ_RAW (c     IN OUT NOCOPY connection,
                  data  IN OUT NOCOPY RAW,
                  len   IN            PLS_INTEGER DEFAULT 1,
                  peek  IN            BOOLEAN     DEFAULT FALSE)
                                      RETURN PLS_INTEGER;
```

Parameters

Table 101–6 read_raw Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The TCP connection to receive data from.
`data (IN OUT COPY)`	The data received.
`len (IN)`	The number of bytes of data to receive.

Table 101–6 read_raw Function Parameters

Parameter	Description
peek (IN)	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.
return value	The actual number of bytes of data received.

Usage Notes

The connection must have already been opened through a call to open_connection(). This function does not return until the specified number of characters have been read, or the end of input has been reached.

If transfer time-out is set when the connection is opened, this function waits for each data packet to be ready to read until time-out occurs. If it occurs, this function stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

Related Functions

read_text(), read_line(), available()

write_raw Function

This function transmits a binary message to a service on an open connection.

Syntax

```
UTL_TCP.WRITE_RAW (c    IN OUT NOCOPY connection,
                   data IN            RAW,
                   len  IN            PLS_INTEGER DEFAULT NULL)
                                      RETURN PLS_INTEGER;
```

Table 101–7 write_raw Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The TCP connection to send data to.
data (IN)	The buffer containing the data to be sent.
len (IN)	The number of bytes of data to transmit. When len is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.
return value	The actual number of bytes of data transmitted.

Usage Notes

The connection must have already been opened through a call to open_connection().

Related Functions

write_text(), write_line(), flush()

read_text Function

This function receives text data from a service on an open connection.

Syntax

```
UTL_TCP.READ_TEXT (c    IN OUT NOCOPY connection,
                   data IN OUT NOCOPY VARCHAR2,
                   len  IN            PLS_INTEGER DEFAULT 1,
                   peek IN            BOOLEAN     DEFAULT FALSE) RETURN PLS_
INTEGER;
```

Table 101–8 read_text Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The TCP connection to receive data from.
data (IN OUT NOCOPY)	The data received.
len (IN)	The number of characters of data to receive.

Table 101–8 read_text Function Parameters

Parameter	Description
peek (IN)	Normally, users want to read the data and remove it from the input queue, that is, consume it. In some situations, users may just want to look ahead at the data without removing it from the input queue so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and an input buffer must be set up when the connection is opened. The amount of data that you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.
return value	The actual number of characters of data received.

Usage Notes

The connection must have already been opened through a call to open_connection(). This function does not return until the specified number of characters has been read, or the end of input has been reached. Text messages will be converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

Unless explicitly overridden, the size of a VARCHAR2 buffer is specified in terms of bytes, while the parameter len refers to the maximum number of characters to be read. When the database character set is multibyte, where a single character may consist of more than 1 byte, you should ensure that the buffer can hold the maximum of characters. In general, the size of the VARCHAR2 buffer should equal the number of characters to be read, multiplied by the maximum number of bytes of a character of the database character set.

If transfer time-out is set when the connection is opened, this function waits for each data packet to be ready to read until time-out occurs. If it occurs, this function stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the partial_multibyte_char exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the read_raw function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time-out occurs, the transfer_timeout

exception is raised instead. The exception can be handled and the read operation can be retried later.

Related Functions

`read_raw(), read_line(), available()`

write_text Function

This function transmits a text message to a service on an open connection.

Syntax

```
UTL_TCP.WRITE_TEXT (c    IN OUT NOCOPY connection,
                    data IN            VARCHAR2,
                    len  IN            PLS_INTEGER DEFAULT NULL)
                                       RETURN PLS_INTEGER;
```

Table 101–9 write_text Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The TCP connection to send data to.
`data (IN)`	The buffer containing the data to be sent.
`len (IN)`	The number of characters of data to transmit. When len is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.
`return value`	The actual number of characters of data transmitted.

Usage Notes

The connection must have already been opened through a call to *open_connection()*. Text messages will be converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.

Related Functions

`write_raw(), write_line(), flush()`

read_line Function

This function receives a text line from a service on an open connection. A line is terminated by a line-feed, a carriage-return or a carriage-return followed by a line-feed.

Syntax

```
UTL_TCP.READ_LINE (c           IN OUT NOCOPY connection,
                   data        IN OUT NOCOPY VARCHAR2,
                   remove_crlf IN            BOOLEAN DEFAULT FALSE,
                   peek        IN            BOOLEAN DEFAULT FALSE)
                                             RETURN PLS_INTEGER;
```

Table 101–10 read_line Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The TCP connection to receive data from.
`data (IN OUT NOCOPY)`	The data received.
`remove_crlf (IN)`	If TRUE, the trailing CR/LF character(s) are removed from the received message.
`peek (IN)`	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.
`return value`	The actual number of characters of data received.

Usage Notes

The connection must have already been opened through a call to *open_connection()*. This function does not return until the end-of-line have been reached, or the end of input has been reached. Text messages will be converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

If transfer time-out is set when the connection is opened, this function waits for each data packet to be ready to read until time-out occurs. If it occurs, this function stops reading and returns all the data read successfully. If no data is read

successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time-out occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

Related Functions

`read_raw(), read_text(), available()`

write_line Function

This function transmits a text line to a service on an open connection. The newline character sequence will be appended to the message before it is transmitted.

Syntax

```
UTL_TCP.WRITE_LINE (c    IN OUT NOCOPY connection,
                    data IN            VARCHAR2 DEFAULT NULL)
                                       RETURN PLS_INTEGER;
```

Table 101–11 write_line Function Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The TCP connection to send data to.
`data (IN)`	The buffer containing the data to be sent.
`return value`	The actual number of characters of data transmitted.

Usage Notes

The connection must have already been opened through a call to *open_connection()*. Text messages will be converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.

Related Functions

`write_raw(), write_text(), flush()`

get_raw(), get_text(), get_line() Functions

Convenient forms of the read functions, which return the data read instead of the amount of data read.

Syntax

```
UTL_TCP.GET_RAW (c     IN OUT NOCOPY connection,
                 len   IN            PLS_INTEGER DEFAULT 1,
                 peek  IN            BOOLEAN     DEFAULT FALSE) RETURN RAW;
UTL_TCP.GET_TEXT (c    IN OUT NOCOPY connection,
                  len  IN            PLS_INTEGER DEFAULT 1,
                  peek IN            BOOLEAN     DEFAULT FALSE) RETURN VARCHAR2;
UTL_TCP.GET_LINE (c           IN OUT NOCOPY connection,
                  remove_crlf IN            BOOLEAN DEFAULT false,
                  peek        IN            BOOLEAN DEFAULT FALSE) RETURN
VARCHAR2;
```

Table 101–12 get_raw(), get_text(), and get_line() Function Parameters

Parameter	Description
c (IN OUT NOCOPY)	The TCP connection to receive data from.
len (IN)	The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek (IN)	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peeked at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf (IN)	If TRUE, the trailing CR/LF character(s) are removed from the received message.

Usage Notes

The connection must have already been opened through a call to *open_connection()*.

For all the get_* APIs described in this section, see the corresponding read_* API for the read time-out issue. For get_text and get_line, see the corresponding read_* API for character set conversion, buffer size, and multibyte character issues.

Related Functions

read_raw(), read_text(), read_line()

flush Procedure

This procedure transmits all data in the output buffer, if a buffer is used, to the server immediately.

Syntax

```
UTL_TCP.FLUSH (c IN OUT NOCOPY connection);
```

Parameters

Table 101–13 flush Procedure Parameters

Parameter	Description
c (IN OUT NOCOPY)	The TCP connection to send data to.

Usage Notes

The connection must have already been opened through a call to *open_connection()*.

Related Functions

write_raw(), write_text(), write_line()

close_connection Procedure

This procedure closes an open TCP/IP connection.

Syntax

```
UTL_TCP.close_CLOSE_CONNECTION (c IN OUT NOCOPY connection);
```

Parameters

Table 101–14 close_connection Procedure Parameters

Parameter	Description
`c (IN OUT NOCOPY)`	The TCP connection to close.

Usage Notes

Connection must have been opened by a previous call to `open_connection()`. The fields `remote_host, remote_port, local_host, local_port` and `charset` of c will be reset after the connection is closed.

An open connection must be closed explicitly. An open connection will remain open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

close_all_connections Procedure

This procedure closes all open TCP/IP connections.

Syntax

```
UTL_TCP.CLOSE_ALL_CONNECTIONS;
```

Usage Notes

This call is provided to close all connections before a PL/SQL program avoid dangling connections.

Related Functions

`open_connection(), close_connection()`

102

UTL_URL

The UTL_URL package has two functions: ESCAPE and UNESCAPE.

See Also: Chapter 96, "UTL_HTTP"

This chapter discusses the following topics:

- Introduction to the UTL_URL Package
- UTL_URL Exceptions
- Summary of UTL_URL Subprograms

Introduction to the UTL_URL Package

A Uniform Resource Locator (URL) is a string that identifies a Web resource, such as a page or a picture. Use a URL to access such resources by way of the HyperText Transfer Protocol (HTTP). For example, the URL for Oracle's Web site is:

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

Normally, a URL contains English alphabetic characters, digits, and punctuation symbols. These characters are known the u*nreserved characters*. Any other characters in URLs, including multibyte characters or binary octet codes, must be escaped to be accurately processed by Web browsers or Web servers. Some punctuation characters, such as dollar sign ($), question mark (?), colon (:), and equals sign (=), are reserved as delimiters in a URL. They are known as the *reserved characters*. To literally process these characters, instead of treating them as delimiters, they must be escaped.

The unreserved characters are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (_), period (.), exclamation point (!), tilde (~), asterisk (*), accent (’), left parenthesis (( ), right parenthesis ( ))

The reserved characters are:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign ($), and comma (,)

The UTL_URL package has two functions that provide escape and unescape mechanisms for URL characters. Use the escape function to escape a URL before the URL is used fetch a Web page by way of the UTL_HTTP package. Use the unescape function to unescape an escaped URL before information is extracted from the URL.

For more information, refer to the Request For Comments (RFC) document RFC2396. Note that this URL escape and unescape mechanism is different from the x-www-form-urlencoded encoding mechanism described in the HTML specification:

```
http://www.w3.org/TR/html
```

You can implement the x-www-form-urlencoded encoding using the UTL_URL.ESCAPE function as follows:

```
CREATE OR REPLACE FUNCTION form_url_encode (
   data IN VARCHAR2,
   charset IN VARCHAR2)
```

```
RETURN VARCHAR2 AS
BEGIN
  RETURN utl_url.escape(data, TRUE, charset); -- note use of TURE
END;
```

For decoding data encoded with the `form-URL-encode scheme`, the following function implements the decording scheme:

```
function form_url_decode(
   data in varchar2,
   charset in varchar2)
  return varchar2 as

begin
  return utl_url.unescape(
     replace(data, '+', ' '),
     charset);
end;
```

UTL_URL Exceptions

Table 102–1 lists the exceptions that can be raised when the UTL_URL package API is invoked.

Table 102–1 UTL_URL Exceptions

Exception	Error Code	Reason
bad_url	29262	The URL contains badly formed escape code sequences
bad_fixed_ width_charset	29274	Fixed-width multibyte character set is not allowed as a URL character set.

Summary of UTL_URL Subprograms

Table 102–2 UTL_URL Package Subprograms

Subprogram	Description
ESCAPE Function on page 102-4	Returns a URL with illegal characters (and optionally reserved characters) escaped using the `%2-digit-hex-code` format
UNESCAPE Function on page 102-6	Unescapes the escape character sequences to their original forms in a URL. Convert the %XX escape character sequences to the original characters

ESCAPE Function

This function returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format.

Syntax

```
UTL_URL.ESCAPE (
   url                   IN VARCHAR2,
   escape_reserved_chars IN BOOLEAN  DEFAULT FALSE,
   url_charset           IN VARCHAR2 DEFAULT
                                     utl_http.body_charset)
   RETURN VARCHAR2;
```

Parameters

Table 102–3 ESCAPE Function Parameters

Parameter	Description
url (IN)	The original URL
escape_reserved_ chars (IN)	Indicates whether the URL reserved characters should be escaped. If set to TRUE, both the reserved and illegal URL characters are escaped. Otherwise, only the illegal URL characters are escaped. The default value is FALSE.
url_charset (IN)	When escaping a character (single-byte or multibyte), what is the target character set that character should be converted to before the character is escaped in %hex-code format? If url_ charset is NULL, the database charset is assumed and no character set conversion will occur. The default value is the current default body character set of the UTL_HTTP package, whose default value is ISO-8859-1. The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.

Usage Notes

Use this function to escape URLs that contain illegal characters as defined in the URL specification RFC 2396. The legal characters in URLs are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (_), period (.), exclamation point (!), tilde (~), asterisk (*), accent (’), left parenthesis (( ), right parenthesis ( ))

The reserved characters consist of:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign ($), and comma (,)

Many of the reserved characters are used as delimiters in the URL. You should escape characters beyond those listed here by using escape_url. Also, to use the reserved characters in the name-value pairs of the query string of a URL, those characters must be escaped separately. An escape_url cannot recognize the need to escape those characters because once inside a URL, those characters become indistinguishable from the actual delimiters. For example, to pass a name-value pair `$logon=scott/tiger` into the query string of a URL, escape the `$` and `/` separately as `%24logon=scott%2Ftiger` and use it in the URL.

Normally, you will escape the entire URL, which contains the reserved characters (delimiters) that should not be escaped. For example:

```
utl_url.escape('http://www.acme.com/a url with space.html')
```

Returns:

```
http://foo.com/a%20url%20with%20space.html
```

In other situations, you may want to send a query string with a value that contains reserved characters. In that case, escape only the value fully (with `escape_reserved_chars` set to `TRUE`) and then concatenate it with the rest of the URL. For example:

```
url := 'http://www.acme.com/search?check=' || utl_url.escape
('Is the use of the "$" sign okay?', TRUE);
```

This expression escapes the question mark (?), dollar sign ($), and space characters in `'Is the use of the "$" sign okay?'` but not the `?` after `search` in the URL that denotes the use of a query string.

The Web server that you intend to fetch Web pages from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be escaped are escaped in the target character set. For example, a user of an EBCDIC database who wants to access an ASCII Web server should escape the URL using `US7ASCII` so that a space is escaped as `%20` (hex code of a space in ASCII) instead of `%40` (hex code of a space in EBCDIC).

This function does not validate a URL for the proper URL format.

UNESCAPE Function

This function unescapes the escape character sequences to its original form in a URL, to convert the %XX escape character sequences to the original characters.

Syntax

```
UTL_URL.UNESCAPE (
   url            IN VARCHAR2,
   url_charset    IN VARCHAR2 DEFAULT utl_http.body_charset)
                  RETURN VARCHAR2;
```

Parameters

Table 102–4 UNESCAPE Function Parameters

Parameter	Description
`url (IN)`	The URL to unescape
`url_charset (IN)`	After a character is unescaped, the character is assumed to be in the `source_charset` character set and it will be converted from the `source_charset` to the database character set before the URL is returned. If `source_charset` is `NULL`, the database charset is assumed and no character set conversion occurred. The default value is the current default body character set of the `UTL_HTTP` package, whose default value is "ISO-8859-1". The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.

Usage Notes

The Web server that you receive the URL from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be unescaped are unescaped in the source character set. For example, a user of an EBCDIC database who receives a URL from an ASCII Web server should unescape the URL using `US7ASCII` so that `%20` is unescaped as a space (0x20 is the hex code of a space in ASCII) instead of a ? (because 0x20 is not a valid character in EBCDIC).

This function does not validate a URL for the proper URL format.

103

ANYDATA TYPE

An ANYDATA contains an instance of a given type, plus a description of the type. In this sense, an ANYDATA is self-describing. An ANYDATA can be persistently stored in the database.

Persistent storage of ANYDATA instances whose type contains embedded LOBs is not supported yet.

This chapter discusses the following topics:

Construction

There are 2 ways to construct an AnyData. The `Convert*()` calls enable construction of the AnyData in its entirety with a single call. They serve as explicit CAST functions from any type in the Oracle ORDBMS to AnyData.

```
STATIC FUNCTION ConvertNumber(num IN NUMBER) RETURN AnyData,
STATIC FUNCTION ConvertDate(dat IN DATE) RETURN AnyData,
STATIC FUNCTION ConvertChar(c IN CHAR) RETURN AnyData,
STATIC FUNCTION ConvertVarchar(c IN VARCHAR) RETURN AnyData,
STATIC FUNCTION ConvertVarchar2(c IN VARCHAR2) RETURN AnyData,
STATIC FUNCTION ConvertRaw(r IN RAW) RETURN AnyData,
STATIC FUNCTION ConvertBlob(b IN BLOB) RETURN AnyData,
STATIC FUNCTION ConvertClob(c IN CLOB) RETURN AnyData,
STATIC FUNCTION ConvertBfile(b IN BFILE) RETURN AnyData,
STATIC FUNCTION ConvertObject(obj IN "<object_type>") RETURN AnyData,
STATIC FUNCTION ConvertRef(rf IN REF "<object_type>") RETURN AnyData,
STATIC FUNCTION ConvertCollection(col IN "<COLLECTION_1>") RETURN AnyData,
```

The second way to construct an AnyData is a piece by piece approach. The `BeginCreate()` call begins the construction process and `EndCreate()` call finishes the construction process. In between these two calls, the individual attributes of an object type or the elements of a collection can be set using `Set*()` calls. For piece by piece access of the attributes of objects and elements of collections, the `PieceWise()` call should be invoked prior to `Get*()` calls.

Note: The AnyData has to be constructed or accessed sequentially starting from its first attribute (or collection element). The `BeginCreate()` call automatically begins the construction in a piece-wise mode. There is no need to call `PieceWise()` immediately after `BeginCreate()`. `EndCreate()` should be called to finish the construction process (before which any access calls can be made).

Summary of ANYDATA Subprograms

Table 103–1 ANYDATA Subprograms

Subprogram	Description
BEGINCREATE Static Procedure on page 103-3	Begins creation process on a new AnyData.
PIECEWISE Member Procedure on page 103-4	Sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

Table 103–1 ANYDATA Subprograms

Subprogram	Description
SET Member Procedures on page 103-4	Sets the current data value.
ENDCREATE Member Procedure on page 103-6	Ends creation of an AnyData.
GETTYPENAME Member Function on page 103-7	Get the fully qualified type name for the AnyData.
GETTYPE Member Function on page 103-7	Gets the Type of the AnyData.
GET Member Functions on page 103-8	Gets the current data value (which should be of appropriate type).

BEGINCREATE Static Procedure

This procedure begins the creation process on a new AnyData.

Syntax

```
STATIC PROCEDURE BeginCreate(
   dtype          IN OUT NOCOPY AnyType,
   adata          OUT NOCOPY AnyData);
```

Parameters

Table 103–2 BEGINCREATE Procedure Parameters

Parameter	Description
dtype	The type of the AnyData. (Should correspond to OCI_TYPECODE_OBJECT or a Collection typecode.)
adata	AnyData being constructed.

Exception

DBMS_TYPES.invalid_parameters: dtype is invalid (not fully constructed, etc.)

Usage Notes

There is no need to call PieceWise() immediately after this call. The construction process begins in a piece-wise manner automatically.

PIECEWISE Member Procedure

This procedure sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of access of the data value to be a collection element at a time (if the data value is of collection type). Once this call has been made, subsequent calls to `Set*()` and `Get*()` will sequentially obtain individual attributes or collection elements.

Syntax

```
MEMBER PROCEDURE PieceWise(
   self         IN OUT NOCOPY AnyData);
```

Parameters

Table 103–3 BEGINCREATE Procedure Parameters

Parameter	Description
`self`	The current data value.

Exceptions

- DBMS_TYPES.invalid_parameters
- DBMS_TYPES.incorrect_usage: On incorrect usage.

Usage Notes

The current data value must be of an OBJECT or COLLECTION type before this call can be made.

Piece-wise construction and access of nested attributes that are of object or collection types is not supported.

SET Member Procedures

Sets the current data value.

This is a list of procedures that should be called depending on the type of the current data value. The type of the data value should be the type of the attribute at the current position during the piece-wise construction process.

Syntax

```
MEMBER PROCEDURE SetNumber(
   self         IN OUT NOCOPY AnyData,
   num          IN NUMBER,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetDate(
   self         IN OUT NOCOPY AnyData,
   dat          IN DATE,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetChar(
   self         IN OUT NOCOPY AnyData,
   c            IN CHAR,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetVarchar(
   self         IN OUT NOCOPY AnyData,
   c            IN VARCHAR,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetVarchar2(
   self         IN OUT NOCOPY AnyData,
   c            IN VARCHAR2,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetRaw(
   self         IN OUT NOCOPY AnyData,
   r            IN RAW,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetBlob(
   self         IN OUT NOCOPY AnyData,
   b            IN BLOB,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetClob(
   self         IN OUT NOCOPY AnyData,
   c            IN CLOB,
   last_elem    IN boolean DEFAULT FALSE)'
MEMBER PROCEDURE SetBfile(
   self         IN OUT NOCOPY AnyData,
   b            IN BFILE,
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetObject(
   self         IN OUT NOCOPY AnyData,
   obj          IN "<object_type>",
   last_elem    IN boolean DEFAULT FALSE);
MEMBER PROCEDURE SetRef(
   self         IN OUT NOCOPY AnyData,
   rf           IN REF "<object_type>",
```

```
   last_elem   IN boolean DEFAULT FALSE),
MEMBER PROCEDURE SetCollection(
   self        IN OUT NOCOPY AnyData,
   col         IN "<collectyion_type>",
   last_elem   IN boolean DEFAULT FALSE);
```

Parameters

Table 103–4 SET*() Procedure Parameters

Parameter	Description
`self`	An AnyData.
`num`	The number, etc., that is to be set.
`last_elem`	Relevant only if AnyData represents a collection. Set to TRUE if it is the last element of the collection, FALSE otherwise.

Exceptions

- DBMS_TYPES.invalid_parameters: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).
- DBMS_TYPES.incorrect_usage: Incorrect usage.
- DBMS_TYPES.type_mismatch: When the expected type is different from the passed in type.

Usage Notes

When `BeginCreate()` is called, construction has already begun in a piece-wise fashion. Subsequent calls to `Set*()` will set the successive attribute values.

If the AnyData is a standalone collection, the `Set*()` call will set the successive collection elements.

ENDCREATE Member Procedure

This procedure ends creation of an AnyData. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE EndCreate(
```

```
   self         IN OUT NOCOPY AnyData);
```

Parameters

Table 103–5 ENDCREATE Procedure Parameter

Parameter	Description
self	An AnyData.

GETTYPENAME Member Function

This function gets the fully qualified type name for the AnyData.

If the AnyData is based on a built-in type, this function will return NUMBER etc.

If it is based on a user defined type, this function will return <schema_ name>.<type_name>. for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

```
MEMBER FUNCTION GetTypeName(
   self         IN AnyData)
   RETURN       VARCHAR2;
```

Parameters

Table 103–6 GETTYPENAME Function Parameter

Parameter	Description
self	An AnyData.

Returns

Type name of the AnyData.

GETTYPE Member Function

This function gets the typecode of the AnyData.

Syntax

```
MEMBER FUNCTION GetType(
```

```
   self          IN AnyData,
   typ           OUT NOCOPY AnyType)
   RETURN        PLS_INTEGER;
```

Parameters

Table 103–7 GETTYPE Function Parameter

Parameter	Description
self	An AnyData.
typ	The AnyType corresponding to the AnyData. May be NULL if it does not represent a user-defined type.

Returns

The typecode corresponding to the type of the AnyData.

GET Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE with which we are accessing (depending on whether we have invoked the `PieceWise()` call).

If `PieceWise()` has NOT been called, we are accessing the AnyData in its entirety and the type of the data value should match the type of the AnyData.

If `PieceWise()` has been called, we are accessing the AnyData piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

Syntax

```
MEMBER FUNCTION GetNumber(
   self          IN AnyData,
   num           OUT NOCOPY NUMBER)
   RETURN        PLS_INTEGER;
  MEMBER FUNCTION GetDate(
   self          IN AnyData,
   dat           OUT NOCOPY DATE)
   RETURN        PLS_INTEGER;
MEMBER FUNCTION GetChar(
   self          IN AnyData,
   c             OUT NOCOPY CHAR)
```

```
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetVarchar(
   self         IN AnyData,
   c            OUT NOCOPY VARCHAR)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetVarchar2(
   self         IN AnyData,
   c            OUT NOCOPY VARCHAR2)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetRaw(
   self         IN AnyData,
   r            OUT NOCOPY RAW)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetBlob(
   self         IN AnyData,
   b            OUT NOCOPY BLOB)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetClob(
   self         IN AnyData,
   c            OUT NOCOPY CLOB)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetBfile(
   self         IN AnyData,
   b            OUT NOCOPY BFILE)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetObject(
   self         IN AnyData,
   obj          OUT NOCOPY "<object_type>")
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetRef(
   self         IN AnyData,
   rf           OUT NOCOPY REF "<object_type>")
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetCollection(
   self         IN AnyData,
   col          OUT NOCOPY "<collection_type>")
   RETURN       PLS_INTEGER;
```

Parameters

Table 103–8 GET* Function Parameter

Parameter	Description
self	An AnyData.

Table 103–8 GET Function Parameter*

Parameter	Description
num	The number, etc., to be obtained.

Returns

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if `PieceWise()` has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.type_mismatch: When the expected type is different from the passed in type.

DBMS_TYPES.invalid_parameters: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.incorrect_usage: Incorrect usage.

104

ANYDATASET TYPE

An ANYDATASET type contains a description of a given type plus a set of data instances of that type. An ANYDATASET can be persistently stored in the database if desired, or it can be used as interface parameters to communicate self-descriptive sets of data, all of which belong to a certain type.

This chapter discusses the following topics:

Construction

The AnyDataSet needs to be constructed value by value, sequentially.

For each data instance (of the type of the AnyDataSet), the `AddInstance()` function must be invoked. This adds a new data instance to the AnyDataSet. Subsequently, `Set*()` can be called to set each value in its entirety.

The MODE of construction / access can be changed to attribute / collection element wise by making calls to `PieceWise()`.

- If the type of the AnyDataSet is TYPECODE_OBJECT, individual attributes will be set with subsequent `Set*()` calls. Likewise on access.
- If the type of the current data value is a collection type individual collection elements will be set with subsequent `Set*()` calls. Likewise on access. This call is very similar to `AnyData.PieceWise()` call defined for the type AnyData.

Note that there is no support for piece-wise construction and access of nested (not top level) attributes that are of object types or collection types.

`EndCreate()` should be called to finish the construction process (before which no access calls can be made).

Summary of ANYDATASET Subprograms

Table 104–1 ANYDATASET Subprograms

Subprogram	Description
BEGINCREATE Static Procedure on page 104-3	The AnyDataSet needs to be constructed value by value, sequentially.
BEGINCREATE Static Procedure on page 104-3	Creates a new AnyDataSet which can be used to create a set of data values of the given ANYTYPE.
ADDINSTANCE Member Procedure on page 104-4	Adds a new data instance to an AnyDataSet.
PIECEWISE Member Procedure on page 104-4	Sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_ OBJECT).
SET* Member Procedures on page 104-5	Sets the current data value.
ENDCREATE Member Procedure on page 104-7	Ends Creation of a AnyDataSet. Other creation functions cannot be called after this call.

Table 104–1 ANYDATASET Subprograms

Subprogram	Description
GETTYPENAME Member Function on page 104-7	Gets the AnyType describing the type of the data instances in an AnyDataSet.
GETTYPE Member Function on page 104-8	Gets the current data value (which should be of appropriate type).
GETINSTANCE Member Function on page 104-9	Gets the next instance in an AnyDataSet.
GET* Member Functions on page 104-9	Gets the current data value (which should be of appropriate type).
GETCOUNT Member Function on page 104-11	Gets the number of data instances in an AnyDataSet.

BEGINCREATE Static Procedure

This procedure creates a new AnyDataSet which can be used to create a set of data values of the given ANYTYPE.

Syntax

```
STATIC PROCEDURE BeginCreate(
   typecode     IN PLS_INTEGER,
   rtype        IN OUT NOCOPY AnyType,
   aset         OUT NOCOPY AnyDataSet);
```

Parameters

Table 104–2 BEGINCREATE Procedure Parameter

Parameter	Description
typecode	The typecode for the type of the AnyDataSet.
dtype	The type of the data values. This parameter is a must for user-defined types like TYPECODE_OBJECT, Collection typecodes, etc.
aset	The AnyDataSet being constructed.

Exceptions

DBMS_TYPES.invalid_parameters: dtype is invalid (not fully constructed, etc.)

ADDINSTANCE Member Procedure

This procedure adds a new data instance to an AnyDataSet.

Syntax

```
MEMBER PROCEDURE AddInstance(
   self          IN OUT NOCOPY AnyDataSet);
```

Parameters

Table 104–3 ADDINSTANCE Procedure Parameter

Parameter	Description
`self`	The AnyDataSet being constructed.

Exceptions

```
DBMS_TYPES.invalid_parameters: Invalid parameters.
DBMS_TYPES.incorrect_usage: On incorrect usage.
```

Usage Notes

The data instances have to be added sequentially. The previous data instance must be fully constructed (or set to NULL) before a new one can be added.

This call DOES NOT automatically set the mode of construction to be piece-wise. The user has to explicitly call `PieceWise()` if a piece-wise construction of the instance is intended.

PIECEWISE Member Procedure

This procedure sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of construction, access of the data value to be a collection element at a time (if the data value is of a collection TYPE). Once this call has been made, subsequent `Set*()` and `Get*()` calls will sequentially obtain individual attributes or collection elements.

Syntax

```
MEMBER PROCEDURE PieceWise(
   self         IN OUT NOCOPY AnyDataSet);
```

Parameters

Table 104–4 PIECEWISE Procedure Parameter

Parameter	Description
self	The AnyDataSet being constructed.

Exceptions

DBMS_TYPES.invalid_parameters
DBMS_TYPES.incorrect_usage: On incorrect usage.

Usage Notes

The current data value must be of an object or collectyon type before this call can be made. There is no support for piece-wise construction or access of embedded object type attributes or nested collections.

SET* Member Procedures

This procedure sets the current data value.

The type of the current data value depends on the MODE with which we are constructing (depending on how we have invoked the PieceWise() call). The type of the current data should be the type of the AnyDataSet if PieceWise() has NOT been called. The type should be the type of the attribute at the current position if PieceWise() has been called.

Syntax

```
MEMBER PROCEDURE SetNumber(
   self              IN OUT NOCOPY AnyDataSet,
   num               IN NUMBER,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetDate(
   self              IN OUT NOCOPY AnyDataSet,
   dat               IN DATE,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetChar(
   self              IN OUT NOCOPY AnyDataSet,
   c                 IN CHAR,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetVarchar(
   self              IN OUT NOCOPY AnyDataSet,
```

```
   c                IN VARCHAR,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetVarchar2(
   self             IN OUT NOCOPY AnyDataSet,
   c                IN VARCHAR2,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetRaw(
   self             IN OUT NOCOPY AnyDataSet,
   r                IN RAW,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetBlob(
   self             IN OUT NOCOPY AnyDataSet,
   b                IN BLOB,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetClob(
   self             IN OUT NOCOPY AnyDataSet,
   c                IN CLOB,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetBfile(
   self             IN OUT NOCOPY AnyDataSet,
   b                IN BFILE,
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetObject(
   self             IN OUT NOCOPY AnyDataSet,
   obj              IN "<object_type>",
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetRef(
   self             IN OUT NOCOPY AnyDataSet,
   rf               IN REF "<object_type>",
   last_elem boolean DEFAULT FALSE);
MEMBER PROCEDURE SetCollection(
   self             IN OUT NOCOPY AnyDataSet,
   col              IN "<collection_type>",
   last_elem boolean DEFAULT FALSE);
```

Parameters

Table 104–5 SET* Procedure Parameters

Parameter	Description
self	The AnyDataSet being accessed.
num	The number, etc., that is to be set.

Table 104–5 SET Procedure Parameters*

Parameter	Description
last_elem	Relevant only if PieceWise() has been already called (for a collection). Set to TRUE if it is the last element of the collection, FALSE otherwise.

Exceptions

- DBMS_TYPES.invalid_parameters: Invalid parameters (if it is not appropriate to add a number at this point in the creation process).
- DBMS_TYPES.incorrect_usage: Incorrect usage.
- DBMS_TYPES.type_mismatch: When the expected type is different from the passed in type.

ENDCREATE Member Procedure

This procedure ends Creation of a AnyDataSet. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE EndCreate(
   self              IN OUT NOCOPY AnyDataSet);
```

Parameters

Table 104–6 ENDCREATE Procedure Parameter

Parameter	Description
self	The AnyDataSet being constructed.

GETTYPENAME Member Function

This procedure gets the fully qualified type name for the AnyDataSet.

If the AnyDataSet is based on a built-in, this function will return NUMBER etc.

If it is based on a user defined type, this function will return <schema_ name>.<type_name>. e.g. SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

```
MEMBER FUNCTION GetTypeName(
   self           IN AnyDataSet)
   RETURN         VARCHAR2;
```

Parameter

Table 104–7 GETTYPENAME Function Parameter

Parameter	Description
self	The AnyDataSet being constructed.

Returns

Type name of the AnyDataSet.

GETTYPE Member Function

Gets the AnyType describing the type of the data instances in an AnyDataSet.

Syntax

```
MEMBER FUNCTION GetType(
   self           IN AnyDataSet,
   typ            OUT NOCOPY AnyType)
   RETURN         PLS_INTEGER;
```

Parameters

Table 104–8 GETTYPE Function Parameter

Parameter	Description
self	The AnyDataSet.
typ	The AnyType corresponding to the AnyData. May be NULL if it does not represent a user-defined function.

Returns

The typecode corresponding to the type of the AnyData.

GETINSTANCE Member Function

This function gets the next instance in an AnyDataSet. Only sequential access to the instances in an AnyDataSet is allowed. After this function has been called, the `Get*()` functions can be invoked on the AnyDataSet to access the current instance. If `PieceWise()` is called before doing the `Get*()` calls, the individual attributes (or collection elements) can be accessed.

It is an error to invoke this function before the AnyDataSet is fully created.

Syntax

```
MEMBER FUNCTION GetInstance(
   self           IN OUT NOCOPY AnyDataSet)
   RETURN         PLS_INTEGER;
```

Parameters

Table 104–9 GETINSTANCE Function Parameter

Parameter	Description
`self`	The `AnyDataSet` being accessed.

Returns

`DBMS_TYPES.SUCCESS` or `DBMS_TYPES.NO_DATA`

`DBMS_TYPES.NO_DATA` signifies the end of the AnyDataSet (all instances have been accessed).

Usage Notes

This function should be called even before accessing the first instance.

GET* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE with which you are accessing it (depending on how we have invoked the `PieceWise()` call). If `PieceWise()` has NOT been called, we are accessing the instance in its entirety and the type of the data value should match the type of the AnyDataSet.

If `PieceWise()` has been called, we are accessing the instance piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

Syntax

```
MEMBER FUNCTION GetNumber(
   self         IN AnyDataSet,
   num          OUT NOCOPY NUMBER)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetDate(
   self         IN AnyDataSet,
   dat          OUT NOCOPY DATE)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetChar(
   self         IN AnyDataSet,
   c            OUT NOCOPY CHAR)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetVarchar(
   self         IN AnyDataSet,
   c            OUT NOCOPY VARCHAR)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetVarchar2(
   self         IN AnyDataSet,
   c            OUT NOCOPY VARCHAR2)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetRaw(
   self         IN AnyDataSet,
   r            OUT NOCOPY RAW)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetBlob(
   self         IN AnyDataSet,
   b            OUT NOCOPY BLOB)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetClob(
   self         IN AnyDataSet,
   c            OUT NOCOPY CLOB)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetBfile(
   self         IN AnyDataSet,
   b            OUT NOCOPY BFILE)
   RETURN       PLS_INTEGER;
MEMBER FUNCTION GetObject(
   self         IN AnyDataSet,
   obj          OUT NOCOPY "<object_type>")
```

```
   RETURN      PLS_INTEGER;
MEMBER FUNCTION GetRef(
   self        IN AnyDataSet,
   rf          OUT NOCOPY REF "<object_type>")
   RETURN      PLS_INTEGER;
MEMBER FUNCTION GetCollection(
   self        IN AnyDataSet,
   col         OUT NOCOPY "<collection_type>")
   RETURN      PLS_INTEGER;
```

Parameters

Table 104–10 GET Procedure Parameters*

Parameter	Description
self	The AnyDataSet being accessed.
num	The number, etc., that is to be obtained.

Returns

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if PieceWise() has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.invalid_parameters: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.incorrect_usage: Incorrect usage

DBMS_TYPES.type_mismatch: When the expected type is different from the passed in type.

GETCOUNT Member Function

This function gets the number of data instances in an AnyDataSet.

Syntax

```
MEMBER FUNCTION GetCount(
   self        IN AnyDataSet)
```

```
    RETURN      PLS_INTEGER;
```

Parameter

Table 104–11 GETCOUNT Function Parameter

Parameter	Description
`self`	The `AnyDataSet` being accessed.

Returns

The number of data instances.

105

ANYTYPE TYPE

An ANYTYPE can contain a type description of any persistent SQL type, named or unnamed, including object types and collection types. It can also be used to construct new transient type descriptions.

New persistent types can only be created using the CREATE TYPE statement. Only new transient types can be constructed using the ANYTYPE interfaces.

Thios chapter discusses the following:

Summary of ANYTYPE Subprograms

Table 105–1 ANYTYPE Subprograms

Subprogram	Description
BEGINCREATE Static Procedure on page 105-2	Creates a new instance of ANYTYPE which can be used to create a transient type description.
SETINFO Member Procedure on page 105-3	Sets any additional information required for constructing a COLLECTION or builtin type.
ADDATTR Member Procedure on page 105-4	Adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPECODE_OBJECT).
ENDCREATE Member Procedure on page 105-5	Ends creation of a transient AnyType. Other creation functions cannot be called after this call.
GETPERSISTENT Static Function on page 105-6	Returns an AnyType corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.
GETINFO Member Function on page 105-6	Gets the type information for the AnyType.
GETATTRELEMINFO Member Function on page 105-8	Gets the type information for an attribute of the type (if it is of TYPECODE_OBJECT). Gets the type information for a collection's element type if the *self* parameter is of a collection type.

BEGINCREATE Static Procedure

This procxedure creates a new instance of ANYTYPE which can be used to create a transient type description.

Syntax

```
STATIC PROCEDURE BEGINCREATE(
   typecode       IN PLS_INTEGER,
   atype          OUT NOCOPY AnyType);
```

Parameters

Table 105–2 BEGINCREATE Procedure Parameters

Parameter	Description
typecode	Use a constant from DBMS_TYPES package. Typecodes for user-defined type: can be DBMS_TYPES.TYPECODE_OBJECT DBMS_TYPES.TYPECODE_VARRAY or DBMS_TYPES.TYPECODE_TABLE Typecodes for builtin types: DBMS_TYPES.TYPECODE_NUMBER etc.
atype	AnyType for a transient type

SETINFO Member Procedure

This procedure sets any additional information required for constructing a COLLECTION or builtin type.

Syntax

```
MEMBER PROCEDURE SetInfo(
   self          IN OUT NOCOPY AnyType,
   prec          IN PLS_INTEGER,
   scale         IN PLS_INTEGER,
   len           IN PLS_INTEGER,
   csid          IN PLS_INTEGER,
   csfrm         IN PLS_INTEGER,
   atype         IN ANYTYPE DEFAULT NULL,
   elem_tc       IN PLS_INTEGER DEFAULT NULL,
   elem_count    IN PLS_INTEGER DEFAULT 0);
```

Parameters

Table 105–3 SETINFO Procedure Parameters

Parameter	Description
self	The transient ANYTYPE that is being constructed.
prec, scale (OPTIONAL)	Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.

Table 105–3 SETINFO Procedure Parameters

Parameter	Description
len (OPTIONAL)	Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
csid, csfrm (OPTIONAL)	Required if typecode represents types requiring character information such as CHAR, VARCHAR, VARCHAR2, or CFILE.
atype (OPTIONAL)	Required if collection element typecode is a user-defined type such as TYPECODE_OBJECT, etc. It is also required for a built-in type that needs user-defined type information such as TYPECODE_REF. This parameter is not needed otherwise.
The Following Parameters Are Required For Collection Types:	
elem_tc	Must be of the collection element's typecode (from DBMS_TYPES package).
elem_count	Pass 0 for elem_count if the self represents a nested table (TYPECODE_TABLE). Otherwise pass the collection count if *self* represents a VARRAY.

Exceptions

- DBMS_TYPES.invalid_parameter: Invalid Parameters (typecode, typeinfo)
- DBMS_TYPES.incorrect_usage: Incorrect usage (cannot call after calling EndCreate(), etc.)

Usage Notes

It is an error to call this function on an AnyType that represents a persistent user defined type.

ADDATTR Member Procedure

This procedure adds an attribute to an AnyType (of typecode DBMS_TYPES.TYPECODE_OBJECT).

Syntax

```
MEMBER PROCEDURE AddAttr(
   self          IN OUT NOCOPY AnyType,
   aname         IN VARCHAR2,
   typecode      IN PLS_INTEGER,
   prec          IN PLS_INTEGER,
```

```
   scale         IN PLS_INTEGER,
   len           IN PLS_INTEGER,
   csid          IN PLS_INTEGER,
   csfrm         IN PLS_INTEGER,
   attr_type     IN ANYTYPE DEFAULT NULL);
```

Parameters

Table 105–4 ADDATTR Procedure Parameters

Parameter	Description
`self`	The transient AnyType that is being constructed. `Must be of type DBMS_TYPES.TYPECODE_OBJECT.`
`aname` (OPTIONAL)	Attribute's name. Could be NULL.
`typecode`	Attribute's typecode. Can be built-in or user-defined typecode (from DBMS_TYPES package).
`prec, scale` (OPTIONAL)	Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
`len` (OPTIONAL)	Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Give length.
`csid, csfrm` (OPTIONAL)	Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, VARCHAR2, CFILE.
`attr_type` (OPTIONAL)	AnyType corresponding to a user-defined type. This parameter is required if the attribute is a user defined type.

Exceptions

- DBMS_TYPES.invalid_parameters: Invalid Parameters (typecode, typeinfo)
- DBMS_TYPES.incorrect_usage: Incorrect usage (cannot call after calling `EndCreate()`, etc.)

ENDCREATE Member Procedure

This procedure ends creation of a transient AnyType. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE EndCreate(
   self           IN OUT NOCOPY AnyType);
```

Parameter

Table 105–5 ENDCREATE Procedure Parameter

Parameter	Description
self	The transient AnyType that is being constructed.

GETPERSISTENT Static Function

This procedure returns an AnyType corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

Syntax

```
STATIC FUNCTION GetPersistent(
   schema_name      IN VARCHAR2,
   type_name        IN VARCHAR2,
   version          IN VARCHAR2 DEFAULT NULL)
   RETURN           AnyType;
```

Parameters

Table 105–6 GETPERSISTENT Function Parameters

Parameter	Description
schema_name	Schema name of the type.
type_name	Type name.
version	Type version.

Returns

An AnyType corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

GETINFO Member Function

This function gets the type information for the AnyType.

Syntax

```
MEMBER FUNCTION GetInfo (
   self        IN AnyType,
```

```
prec        OUT PLS_INTEGER,
scale       OUT PLS_INTEGER,
len         OUT PLS_INTEGER,
csid        OUT PLS_INTEGER,
csfrm       OUT PLS_INTEGER,
schema_name OUT VARCHAR2,
type_name   OUT VARCHAR2,
version     OUT varchar2,
count       OUT PLS_INTEGER)
RETURN      PLS_INTEGER;
```

Parameters

Table 105–7 GETINFO Function Parameters

Parameter	Description
`self`	The AnyType.
`prec, scale`	If typecode represents a number. Gives precision and scale. Ignored otherwise.
`len`	If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
`csid, csfrm`	If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE.
`schema_name, type_ name, version`	Type's schema (if persistent), typename and version.
`count`	If *self* is a VARRAY, this gives the VARRAY count. If self is of TYPECODE_OBJECT, this gives the number of attributes.

Returns

The typecode of *self*.

Exceptions

- DBMS_TYPES.invalid_parameters: Invalid Parameters (position is beyond bounds or the AnyType is not properly Constructed).

GETATTRELEMINFO Member Function

This function gets the type information for an attribute of the type (if it is of TYPECODE_OBJECT). Gets the type information for a collection's element type if the *self* parameter is of a collection type.

Syntax

```
MEMBER FUNCTION GetAttrElemInfo (
   self           IN AnyType,
   pos            IN PLS_INTEGER,
   prec           OUT PLS_INTEGER,
   scale          OUT PLS_INTEGER,
   len            OUT PLS_INTEGER,
   csid           OUT PLS_INTEGER,
   csfrm          OUT PLS_INTEGER,
   attr_elt_type  OUT ANYTYPE
   aname          OUT VARRCHAR2)
   RETURN         PLS_INTEGER;
```

Parameters

Table 105–8 GETATTRELEMINFO Function Parameters

Parameter	Description
`self`	The AnyType.
`pos`	If self is of TYPECODE_OBJECT, this gives the attribute position (starting at 1). It is ignored otherwise.
`prec, scale`	If attribute / collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.
`len`	If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
`csid, csfrm`	If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE. Gives character set ID, character set form.
`attr_elt_type`	IF attribute / collection element typecode represents a user-defined type, this returns the AnyType corresponding to it. User can subsequently describe the *attr_elt_type*.
`aname`	Attribute name (if it is an attribute of an object type, NULL otherwise).

Returns

The typecode of the attribute or collection element.

Exceptions

- DBMS_TYPES.invalid_parameters: Invalid Parameters (position is beyond bounds or the AnyType is not properly constructed).

106

Advanced Queuing Types

This chapter describes the types designed for use with the following Advanced Queuing (AQ) packages:

- DBMS_AQ
- DBMS_AQADM

This chapter describes the following topics:

- Advanced Queuing Types

See Also:

- *Oracle9i Application Developer's Guide - Advanced Queuing* contains information about using AQ
- *Oracle9i Supplied PL/SQL Packages and Types Reference* contains information about the DBMS_AQ and DBMS_AQADM packages

Advanced Queuing Types

Table 106–1 Advanced Queuing Types

Type	Description
AQ$_AGENT Type on page 106-2	Identifies a producer or a consumer of a message
AQ$_AGENT_LIST_T Type on page 106-3	Identifies the list of agents for which DBMS_AQ.LISTEN listens
AQ$_DESCRIPTOR Type on page 106-3	Specifies the AQ descriptor received by the AQ PL/SQL callbacks upon notification

Table 106–1 (Cont.) Advanced Queuing Types

Type	Description
AQ$_POST_INFO Type on page 106-4	Specifies anonymous subscriptions to which you want to post messages
AQ$_POST_INFO_LIST Type on page 106-4	Identifies the list of anonymous subscriptions to which you want to post messages
AQ$_RECIPIENT_LIST_T Type on page 106-5	Identifies the list of agents that receive the message
AQ$_REG_INFO Type on page 106-5	Identifies a producer or a consumer of a message
AQ$_REG_INFO_LIST Type on page 106-7	Identifies the list of registrations to a queue
AQ$_SUBSCRIBER_LIST_T Type on page 106-7	Identifies the list of subscribers that subscribe to a queue
DEQUEUE_OPTIONS_T Type on page 106-8	Specifies the options available for the dequeue operation
ENQUEUE_OPTIONS_T Type on page 106-10	Specifies the options available for the enqueue operation
MESSAGE_PROPERTIES_T Type on page 106-11	Describes the information that is used by AQ to manage individual messages

AQ$_AGENT Type

Identifies a producer or a consumer of a message.

Syntax

```
TYPE SYS.AQ$_AGENT IS OBJECT (
   name       VARCHAR2(30),
   address    VARCHAR2(1024),
   protocol   NUMBER  DEFAULT 0);
```

Attributes

Table 106–2 AQ$_AGENT Attributes

Attribute	Description
name	Name of a producer or consumer of a message. The name must follow object name guidelines in the *Oracle9i SQL Reference* with regard to reserved characters.
address	Protocol-specific address of the recipient. If the protocol is 0, then the address is of the form [*schema.*]*queue*[@*dblink*]. For example, a queue named emp_messages in the hr queue at the site dbs1.net has the following address: hr.emp_messages@dbs1.net
protocol	Protocol to interpret the address and propagate the message.

AQ$_AGENT_LIST_T Type

Identifies the list of agents for which DBMS_AQ.LISTEN listens.

See Also: "AQ$_AGENT Type" on page 106-2

Syntax

```
TYPE SYS.AQ$_AGENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY INTEGER;
```

AQ$_DESCRIPTOR Type

Specifies the AQ descriptor received by the AQ PL/SQL callbacks upon notification.

See Also: "MESSAGE_PROPERTIES_T Type" on page 106-11

Syntax

```
TYPE SYS.AQ$_DESCRIPTOR IS OBJECT (
   queue_name      VARCHAR2(61),
   consumer_name   VARCHAR2(30),
   msg_id          RAW(16),
   msg_prop        MSG_PROP_T);
```

Attributes

Table 106–3 AQ$_DESCRIPTOR Attributes

Attribute	Description
queue_name	Name of the queue in which the message was enqueued which resulted in the notification
consumer_name	Name of the consumer for the multiconsumer queue
msg_id	Identification number of the message
msg_prop	Message properties specified by the MSG_PROP_T type

AQ$_POST_INFO Type

Specifies anonymous subscriptions to which you want to post messages.

Syntax

```
TYPE SYS.AQ$_POST_INFO IS OBJECT (
   name        VARCHAR2(128),
   namespace   NUMBER,
   payload     RAW(2000)  DEFAULT NULL);
```

Attributes

Table 106–4 AQ$_POST_INFO Attributes

Attribute	Description
name	The name of the anonymous subscription to which you want to post
namespace	To receive notifications from other applications through DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS
payload	The payload to be posted to the anonymous subscription

AQ$_POST_INFO_LIST Type

Identifies the list of anonymous subscriptions to which you want to post messages.

See Also: "AQ$_POST_INFO Type" on page 106-4

Syntax

```
TYPE SYS.AQ$_POST_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_POST_INFO;
```

AQ$_RECIPIENT_LIST_T Type

Identifies the list of agents that receive the message. This type can be used only when the queue is enabled for multiple dequeues.

See Also: "AQ$_AGENT Type" on page 106-2

Syntax

```
TYPE SYS.AQ$_RECIPIENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;
```

AQ$_REG_INFO Type

Specifies the information regarding the registrant for notification on a queue.

Syntax

```
TYPE SYS.AQ$_REG_INFO IS OBJECT (
   name        VARCHAR2(128),
   namespace   NUMBER,
   callback    VARCHAR2(4000),
   context     RAW(2000)  DEFAULT NULL);
```

Attributes

Table 106–5 AQ Registration Info Type Attributes

Attribute	Description
name	Specifies the name of the subscription. The subscription name is of the form *schema.queue* if the registration is for a single consumer queue or *schema.queue:consumer_name* if the registration is for a multiconsumer queues.

Table 106–5 (Cont.) AQ Registration Info Type Attributes

Attribute	Description
namespace	Specifies the namespace of the subscription. To receive notifications from AQ queues, the namespace must be DBMS_AQ.NAMESPACE_AQ. To receive notifications from other applications through DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.
callback	Specifies the action to be performed on message notification. For HTTP notifications, the form is the following: http://www.company.com:8080 For e-mail notifications, the form is the following: mailto://xyz@company.com For raw message payload for the PLSQLCALLBACK procedure, use the following: plsql://*schema.procedure*?PR=0 For user-defined type message payload converted to XML for the PLSQLCALLBACK procedure, use the following: plsql://*schema.procedure*?PR=1
context	Specifies the context that is to be passed to the callback function.

Usage Notes

You can use the following notification mechanisms: OCI, e-mail, or PL/SQL callback. Notification for nonpersistent queues depends on the notification mechanism and the queue payload type specified, as shown in Table 106–6.

Table 106–6 Nonpersistent Queues

Queue Payload Type	Presentation Specified					
	RAW			XML		
	Notification Mechanism			Notification Mechanism		
	OCI	E-mail	PL/SQL Callback	OCI	E-mail	PL/SQL Callback
RAW	The callback receives the RAW data in the payload.	Not supported	The PL/SQL callback receives the RAW data in the payload.	The callback receives the XML data in the payload.	The XML data is formatted as an IDAP message and e-mailed to the registered e-mail address.	The PL/SQL callback receives the XML data in the payload.
ADT	Not supported.	Not supported.	Not supported.	The callback receives the XML data in the payload.	The XML data is formatted as an IDAP message and e-mailed to the registered e-mail address.	The PL/SQL callback receives the XML data in the payload.

AQ$_REG_INFO_LIST Type

Identifies the list of registrations to a queue.

See Also: "AQ$_REG_INFO Type" on page 106-5

Syntax

```
TYPE SYS.AQ$_REG_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_REG_INFO;
```

AQ$_SUBSCRIBER_LIST_T Type

Identifies the list of subscribers that subscribe to a queue.

See Also: "AQ$_AGENT Type" on page 106-2

Syntax

```
TYPE SYS.AQ$_SUBSCRIBER_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;
```

DEQUEUE_OPTIONS_T Type

Specifies the options available for the dequeue operation.

Syntax

```
TYPE DEQUEUE_OPTIONS_T IS RECORD (
   consumer_name     VARCHAR2(30)    DEFAULT NULL,
   dequeue_mode      BINARY_INTEGER  DEFAULT REMOVE,
   navigation        BINARY_INTEGER  DEFAULT NEXT_MESSAGE,
   visibility        BINARY_INTEGER  DEFAULT ON_COMMIT,
   wait              BINARY_INTEGER  DEFAULT FOREVER,
   msgid             RAW(16)         DEFAULT NULL,
   correlation       VARCHAR2(128)   DEFAULT NULL,
   deq_condition     VARCHAR2(4000)  DEFAULT NULL,
   transformation    VARCHAR2(60)    DEFAULT NULL);
```

Attributes

Table 106–7 DEQUEUE_OPTIONS_T Attributes

Attribute	Description
consumer_name	Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should be set to NULL. For secure queues, consumer_name must be a valid AQ Agent
dequeue_mode	Specifies the locking behavior associated with the dequeue. The possible settings follow: BROWSE: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement. LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement. REMOVE: Read the message and update or delete it. This setting is the default. The message can be retained in the queue table based on the retention properties. REMOVE_NODATA: Mark the message as updated or deleted. The message can be retained in the queue table based on the retention properties.

Table 106–7 (Cont.) DEQUEUE_OPTIONS_T Attributes

Attribute	Description
navigation	Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. The possible settings follow: NEXT_MESSAGE: Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message that matches the search criteria and belongs to the message group. This setting is the default. NEXT_TRANSACTION: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This setting can only be used if message grouping is enabled for the current queue. FIRST_MESSAGE: Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue.
visibility	Specifies whether the new message is dequeued as part of the current transaction.The visibility parameter is ignored when using the BROWSE dequeue mode. The possible settings follow: ON_COMMIT: The dequeue will be part of the current transaction. This setting is the default. IMMEDIATE: The dequeued message is not part of the current transaction. It constitutes a transaction on its own.
wait	Specifies the wait time if there is currently no message available which matches the search criteria. The possible settings follow: FOREVER: Wait forever. This setting is the default. NO_WAIT: Do not wait. number: Wait time in seconds.
msgid	Specifies the message identifier of the message to be dequeued.
correlation	Specifies the correlation identifier of the message to be dequeued. Special pattern matching characters, such as the percent sign (%) and the underscore (_) can be used. If more than one message satisfies the pattern, then the order of dequeuing is undetermined.

Table 106–7 (Cont.) DEQUEUE_OPTIONS_T Attributes

Attribute	Description
deq_condition	A conditional expression based on the message properties, the message data properties, and PL/SQL functions. A deq_condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the WHERE clause of a SQL query). Message properties include priority, corrid and other columns in the queue table To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The deq_condition parameter cannot exceed 4000 characters.
transformation	Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.

ENQUEUE_OPTIONS_T Type

Specifies the options available for the enqueue operation.

Syntax

```
TYPE SYS.ENQUEUE_OPTIONS_T IS RECORD (
   visibility            BINARY_INTEGER  DEFAULT ON_COMMIT,
   relative_msgid        RAW(16)         DEFAULT NULL,
   sequence_deviation    BINARY_INTEGER  DEFAULT NULL,
   transformation        VARCHAR2(60)    DEFAULT NULL);
```

Attributes

Table 106–8 ENQUEUE_OPTIONS_T Attributes

Attribute	Description
visibility	Specifies the transactional behavior of the enqueue request. The possible settings follow: ON_COMMIT: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This setting is the default. IMMEDIATE: The enqueue is not part of the current transaction. The operation constitutes a transaction on its own. This is the only value allowed when enqueuing to a non-persistent queue.
relative_msgid	Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid only if BEFORE is specified in sequence_deviation. This parameter is ignored if sequence deviation is not specified.
sequence_deviation	Specifies whether the message being enqueued should be dequeued before other messages already in the queue. The possible settings follow: BEFORE: The message is enqueued ahead of the message specified by relative_msgid. TOP: The message is enqueued ahead of any other messages.
transformation	Specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.

MESSAGE_PROPERTIES_T Type

Describes the information that AQ uses to manage individual messages. These are set at enqueue time, and their values are returned at dequeue time.

See Also: "AQ$_RECIPIENT_LIST_T Type" on page 106-5

Syntax

```
TYPE MESSAGE_PROPERTIES_T IS RECORD (
   priority               BINARY_INTEGER  DEFAULT 1,
   delay                  BINARY_INTEGER  DEFAULT NO_DELAY,
   expiration             BINARY_INTEGER  DEFAULT NEVER,
   correlation            VARCHAR2(128)   DEFAULT NULL,
   attempts               BINARY_INTEGER,
```

```
recipient_list          AQ$_RECIPIENT_LIST_T,
exception_queue         VARCHAR2(51)    DEFAULT NULL,
enqueue_time            DATE,
state                   BINARY_INTEGER,
sender_id               AQ$_AGENT       DEFAULT NULL,
original_msgid          RAW(16)         DEFAULT NULL);
```

Attributes

Table 106–9 MESSAGE_PROPERTIES_T Attributes

Attribute	Description
`priority`	Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
`delay`	Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by `msgid` overrides the delay specification. A message enqueued with delay set is in the `WAITING` state, and when the delay expires, the message goes to the `READY` state. `DELAY` processing requires the queue monitor to be started. Delay is set by the producer who enqueues the message. The possible settings follow: `NO_DELAY`: The message is available for immediate dequeuing number: The number of seconds to delay the message
`expiration`	Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the delay. Expiration processing requires the queue monitor to be running. The possible settings follow: `NEVER`: The message does not expire number: The number of seconds message remains in `READY` state. If the message is not dequeued before it expires, then it is moved to the exception queue in the `EXPIRED` state.
`correlation`	Returns the identification supplied by the producer for a message at enqueuing
`attempts`	Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.

Table 106–9 (Cont.) MESSAGE_PROPERTIES_T Attributes

Attribute	Description
recipient_list	This parameter is only valid for queues that allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time. For type definition, see the "AQ$_AGENT Type" on page 106-2.
exception_queue	Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved automatically in the following cases: ▪ The number of unsuccessful dequeue attempts has exceeded the specification for the max_retries parameter in the DBMS_AQADM.CREATE_QUEUE procedure during queue creation. You can view the max_retries for a queue in the ALL_QUEUES data dictionary view. ▪ All messages in the exception queue are in the EXPIRED state. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert file. If the default exception queue is specified, then the parameter returns a NULL value at dequeue time.
enqueue_time	Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user. This parameter cannot be set at enqueue time.
state	Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow: 0: The message is ready to be processed. 1: The message delay has not yet been reached. 2: The message has been processed and is retained. 3: The message has been moved to the exception queue.
sender_id	Specifies the application-specified sender identification. You must specify sender_id to enqueue messages to secure queues.
original_msgid	This parameter is used by Oracle AQ for propagating messages.

107

JMS Types

In release 9.2, member procedures and functions and static functions are added to existing JMS PL/SQL types (`aq$_jms_text_message` and `aq$_jms_bytes_message`) so that a PL/SQL application can use JMS queues of these types. The aq$_jms_message type is added so that JMS messages of different types can be enqueued on the same AQ queue. Using the release 9.2 member procedures and functions, messages enqueued from PL/SQL can be dequeued in OJMS, and messages enqueued from OJMS can be dequeued in PL/SQL.

> **See Also:** *Oracle9i Application Developer's Guide - Advanced Queuing*

This chapter discusses the following topics:

Constants to Support the aq$_jms_message Type

These constants are part of the DBMS_AQJMS package.

- `JMS_TEXT_MESSAGE   CONSTANT BINARY_INTEGER;`
- `JMS_BYTES_MESSAGE  CONSTANT BINARY_INTEGER;`
- `JMS_STREAM_MESSAGE CONSTANT BINARY_INTEGER;`
- `JMS_MAP_MESSAGE    CONSTANT BINARY_INTEGER;`
- `JMS_OBJECT_MESSAGE CONSTANT BINARY_INTEGER;`

See "aq$_jms_message Type" on page 107-4 for more information.

Summary of JMS Types

This chapter discusses the following JMS types:

- aq$_jms_userproperty Type
- aq$_jms_userproparray Type
- aq$_jms_header Type
- aq$_jms_message Type
- aq$_jms_text_message Type
- aq$_jms_bytes_message Type

aq$_jms_userproperty Type

This type is used to store an individual user-specified JMS message user property.

Syntax

```
TYPE aq$_jms_userproperty AS object(
   name        VARCHAR(100),
   type        INT,
   str_value   VARCHAR(2000),
   num_value   NUMBER,
   java_type   INT);
```

aq$_jms_userproparray Type

This type is used to store the list of JMS user-specified message properties for a given JMS message.

Syntax

```
TYPE aq$_jms_userproparray AS varray(100) of aq$_jms_userproperty;
```

aq$_jms_header Type

This type is used to store the JMS message header values for a given JMS message.

Syntax

```
TYPE aq$_jms_header AS object(
   replyto     sys.aq$_agent,
   type        VARCHAR(100),
   userid      VARCHAR(100),
   appid       VARCHAR(100),
   groupid     VARCHAR(100),
   groupseq    INT,
   properties  aq$_jms_userproparray,
   MEMBER PROCEDURE lookup_property_name (new_property_name IN VARCHAR),
   MEMBER PROCEDURE set_replyto          (replyto           IN sys.aq$_agent),
   MEMBER PROCEDURE set_type             (type              IN VARCHAR),
   MEMBER PROCEDURE set_userid           (userid            IN VARCHAR),
   MEMBER PROCEDURE set_appid            (appid             IN VARCHAR),
   MEMBER PROCEDURE set_groupid          (groupid           IN VARCHAR),
   MEMBER PROCEDURE set_groupseq         (groupseq          IN INT),
   MEMBER PROCEDURE clear_properties,
   MEMBER PROCEDURE set_boolean_property(
      property_name   IN  VARCHAR,
      property_value  IN  BOOLEAN),
   MEMBER PROCEDURE set_byte_property(
     property_name   IN  VARCHAR,
     property_value  IN  INT ),
   MEMBER PROCEDURE set_short_property (
     property_name   IN  VARCHAR,
     property_value  IN  INT ),
   MEMBER PROCEDURE set_int_property (
      property_name   IN  VARCHAR,
      property_value  IN  INT ),
   MEMBER PROCEDURE set_long_property (
      property_name   IN  VARCHAR,
```

```
    property_value  IN  NUMBER ),
MEMBER PROCEDURE set_float_property (
    property_name   IN  VARCHAR,
    property_value  IN  FLOAT ),
MEMBER PROCEDURE set_double_property (
    property_name   IN  VARCHAR,
    property_value  IN  DOUBLE PRECISION ),
MEMBER PROCEDURE set_string_property (
    property_name   IN  VARCHAR,
    property_value  IN  VARCHAR ),
MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
MEMBER FUNCTION get_type     RETURN VARCHAR,
MEMBER FUNCTION get_userid   RETURN VARCHAR,
MEMBER FUNCTION get_appid    RETURN VARCHAR,
MEMBER FUNCTION get_groupid  RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER FUNCTION get_boolean_property (property_name IN  VARCHAR)
  RETURN   BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN  VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_short_property   (property_name IN  VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_int_property     (property_name IN  VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_long_property    (property_name IN  VARCHAR)
  RETURN   NUMBER,
MEMBER FUNCTION get_float_property   (property_name IN  VARCHAR)
  RETURN   FLOAT,
MEMBER FUNCTION get_double_property  (property_name IN  VARCHAR)
  RETURN   DOUBLE PRECISION,
MEMBER FUNCTION get_string_property  (property_name IN  VARCHAR)
  RETURN   VARCHAR);
```

aq$_jms_message Type

This type is the ADT used to store JMS messages of all the JMS -specified message types: `JMSText`, `JMSBytes`, `JMSMap`, `JMSStream`, and `JMSObject`.

The static function contruct defined as a part of `aq$_jms_message` is

`STATIC FUNCTION construct ( mtype IN int ) RETURN aq$_jms_message.`

See "CONSTRUCT Static Function" on page 107-27 for more information.

Syntax

```
TYPE aq$_jms_message AS object(
   header        aq$_jms_header,
   senderid      varchar2(100),
   message_type  INT,
   text_len      INT,
   bytes_len     INT,
   text_vc       varchar2(4000),
   bytes_raw     raw(2000),
   text_lob      clob,
   bytes_lob     blob,
   STATIC FUNCTION construct     (mtype    IN  INT) RETURN aq$_jms_message,
   MEMBER PROCEDURE set_text     (payload  IN  VARCHAR2),
   MEMBER PROCEDURE set_text     (payload  IN  CLOB),
   MEMBER PROCEDURE get_text     (payload  OUT VARCHAR2),
   MEMBER PROCEDURE get_text     (payload  OUT CLOB),
   MEMBER PROCEDURE set_bytes    (payload  IN  RAW),
   MEMBER PROCEDURE set_bytes    (payload  IN  BLOB),
   MEMBER PROCEDURE get_bytes    (payload  OUT RAW),
   MEMBER PROCEDURE get_bytes    (payload  OUT BLOB),
   MEMBER PROCEDURE set_replyto  (replyto  IN  sys.aq$_agent),
   MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
   MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
   MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
   MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
   MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
   MEMBER PROCEDURE clear_properties ,
   MEMBER PROCEDURE set_boolean_property(
      property_name   IN      VARCHAR,
      property_value  IN      BOOLEAN),
   MEMBER PROCEDURE set_byte_property(
      property_name   IN      VARCHAR,
      property_value  IN      INT),
   MEMBER PROCEDURE set_short_property(
      property_name   IN      VARCHAR,
      property_value  IN      INT),
   MEMBER PROCEDURE set_int_property(
      property_name   IN      VARCHAR,
      property_value  IN      INT),
  MEMBER PROCEDURE set_long_property(
     property_name   IN      VARCHAR,
     property_value  IN      NUMBER),
  MEMBER PROCEDURE set_float_property(
     property_name   IN      VARCHAR,
```

```
   property_value  IN      FLOAT),
  MEMBER PROCEDURE set_double_property(
   property_name   IN      VARCHAR,
   property_value  IN      DOUBLE PRECISION),
  MEMBER PROCEDURE set_string_property(
   property_name   IN      VARCHAR,
   property_value  IN      VARCHAR),
  MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
  MEMBER FUNCTION get_type     RETURN VARCHAR,
  MEMBER FUNCTION get_userid   RETURN VARCHAR,
  MEMBER FUNCTION get_appid    RETURN VARCHAR,
  MEMBER FUNCTION get_groupid  RETURN VARCHAR,
  MEMBER FUNCTION get_groupseq RETURN INT,
  MEMBER FUNCTION get_boolean_property (property_name   IN   VARCHAR)
    RETURN   BOOLEAN,
  MEMBER FUNCTION get_byte_property    (property_name   IN   VARCHAR)
    RETURN   INT,
  MEMBER FUNCTION get_short_property   (property_name   IN   VARCHAR)
    RETURN   INT,
  MEMBER FUNCTION get_int_property     (property_name   IN   VARCHAR)
    RETURN   INT,
  MEMBER FUNCTION get_long_property    (property_name   IN   VARCHAR)
    RETURN   NUMBER,
  MEMBER FUNCTION get_float_property   (property_name   IN   VARCHAR)
    RETURN   FLOAT,
  MEMBER FUNCTION get_double_property  (property_name   IN   VARCHAR)
    RETURN   DOUBLE PRECISION,
  MEMBER FUNCTION get_string_property  (property_name   IN   VARCHAR)
    RETURN   VARCHAR);
```

aq$_jms_text_message Type

This type is the ADT used to store a `JMSText` message in an AQ queue.

Syntax

```
TYPE aq$_jms_text_message AS object(
  header    aq$_jms_header,
  text_len  INT,
  text_vc   varchar2(4000),
  text_lob  clob,
  STATIC FUNCTION construct RETURN aq$_jms_text_message,
  MEMBER PROCEDURE set_text     (payload  IN  VARCHAR2),
  MEMBER PROCEDURE set_text     (payload  IN  CLOB),
  MEMBER PROCEDURE get_text     (payload  OUT VARCHAR2),
```

```
MEMBER PROCEDURE get_text     (payload  OUT CLOB),
MEMBER PROCEDURE set_replyto  (replyto  IN  sys.aq$_agent),
MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
MEMBER PROCEDURE clear_properties,
MEMBER PROCEDURE set_boolean_property(
   property_name   IN      VARCHAR,
   property_value  IN      BOOLEAN),
MEMBER PROCEDURE set_byte_property (
   property_name   IN      VARCHAR,
   property_value  IN      INT ),
MEMBER PROCEDURE set_short_property (
   property_name   IN      VARCHAR,
   property_value  IN      INT ),
MEMBER PROCEDURE set_int_property (
   property_name   IN      VARCHAR,
   property_value  IN      INT ),
MEMBER PROCEDURE set_long_property (
   property_name   IN      VARCHAR,
   property_value  IN      NUMBER ),
MEMBER PROCEDURE set_float_property (
   property_name   IN      VARCHAR,
   property_value  IN      FLOAT ),
MEMBER PROCEDURE set_double_property (
   property_name   IN      VARCHAR,
   property_value  IN      DOUBLE PRECISION ),
MEMBER PROCEDURE set_string_property (
   property_name   IN      VARCHAR,
   property_value  IN      VARCHAR ),
MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
MEMBER FUNCTION get_type     RETURN VARCHAR,
MEMBER FUNCTION get_userid   RETURN VARCHAR,
MEMBER FUNCTION get_appid    RETURN VARCHAR,
MEMBER FUNCTION get_groupid  RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER FUNCTION get_boolean_property (property_name IN  VARCHAR)
  RETURN   BOOLEAN,
MEMBER FUNCTION get_byte_property   (property_name IN  VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_short_property  (property_name IN  VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_int_property    (property_name IN  VARCHAR)
```

```
  RETURN   INT,
MEMBER FUNCTION get_long_property   (property_name IN  VARCHAR)
  RETURN   NUMBER,
MEMBER FUNCTION get_float_property  (property_name IN  VARCHAR)
  RETURN   FLOAT,
MEMBER FUNCTION get_double_property (property_name IN  VARCHAR)
  RETURN   DOUBLE PRECISION,
MEMBER FUNCTION get_string_property (property_name IN  VARCHAR)
  RETURN   VARCHAR);
```

aq$_jms_bytes_message Type

This type is the ADT used to store a JMSBytes message in an AQ queue.

Syntax

```
TYPE aq$_jms_bytes_message AS object(
  header     aq$_jms_header,
  bytes_len  INT,
  bytes_raw  raw(2000),
  bytes_lob  blob,
  STATIC FUNCTION construct RETURN aq$_jms_bytes_message,
  MEMBER PROCEDURE set_bytes    (payload  IN RAW),
  MEMBER PROCEDURE set_bytes    (payload  IN BLOB),
  MEMBER PROCEDURE get_bytes    (payload  OUT RAW),
  MEMBER PROCEDURE get_bytes    (payload  OUT BLOB),
  MEMBER PROCEDURE set_replyto  (replyto  IN  sys.aq$_agent),
  MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
  MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
  MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
  MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
  MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
  MEMBER PROCEDURE clear_properties,
  MEMBER PROCEDURE set_boolean_property(
     property_name   IN  VARCHAR,
     property_value  IN  BOOLEAN),
  MEMBER PROCEDURE set_byte_property(
     property_name   IN  VARCHAR,
     property_value  IN  INT),
  MEMBER PROCEDURE set_short_property(
     property_name   IN  VARCHAR,
                 property_value  IN INT),
  MEMBER PROCEDURE set_int_property(
                 property_name   IN VARCHAR,
                 property_value  IN INT),
```

```
MEMBER PROCEDURE set_long_property(
              property_name   IN VARCHAR,
              property_value  IN NUMBER),
MEMBER PROCEDURE set_float_property(
              property_name   IN VARCHAR,
              property_value  IN FLOAT),
MEMBER PROCEDURE set_double_property(
              property_name   IN VARCHAR,
              property_value  IN DOUBLE PRECISION),
MEMBER PROCEDURE set_string_property(
              property_name   IN VARCHAR,
              property_value  IN VARCHAR),
MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
MEMBER FUNCTION get_type RETURN VARCHAR,
MEMBER FUNCTION get_userid RETURN VARCHAR,
MEMBER FUNCTION get_appid RETURN VARCHAR,
MEMBER FUNCTION get_groupid RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER FUNCTION get_boolean_property (property_name IN   VARCHAR)
  RETURN   BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN   VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_short_property   (property_name IN   VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_int_property     (property_name IN   VARCHAR)
  RETURN   INT,
MEMBER FUNCTION get_long_property    (property_name IN   VARCHAR)
  RETURN   NUMBER,
MEMBER FUNCTION get_float_property   (property_name IN   VARCHAR)
  RETURN   FLOAT,
MEMBER FUNCTION get_double_property  (property_name IN   VARCHAR)
  RETURN   DOUBLE PRECISION,
MEMBER FUNCTION get_string_property  (property_name IN   VARCHAR)
  RETURN   VARCHAR);
```

Summary of JMS Type Member and Static Subprograms

Table 107–1 JMS Types—Member and Static Subprograms

Subprogram	Description
LOOKUP_PROPERTY_NAME Member Procedure	Checks whether `new_property_name` exists in the properties

Table 107–1 (Cont.) JMS Types—Member and Static Subprograms

Subprogram	Description
SET_REPLYTO Member Procedure	Sets the `replyto` parameter, which corresponds to `JMSReplyTo`
SET_TYPE Member Procedure	Sets the JMS type, which can be any text, and which corresponds to `JMSType`
SET_USERID Member Procedure	Sets `userid`, which corresponds to `JMSXUserID`
SET_APPID Member Procedure	Sets `appid`, which corresponds to `JMSXAppID`
SET_GROUPID Member Procedure	Sets `groupid`, which corresponds to `JMSXGroupID`
SET_GROUPSEQ Member Procedure	Sets `groupseq`, which corresponds to `JMSXGroupSeq`
CLEAR_PROPERTIES Member Procedure	Clears all properties
SET_BOOLEAN_ PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_BYTE_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_SHORT_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_INT_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_LONG_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_FLOAT_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_DOUBLE_ PROPERTY Member Procedure	Checks whether `property_name` is null or exists
SET_STRING_PROPERTY Member Procedure	Checks whether `property_name` is null or exists
GET_REPLYTO Member Function	Returns `replyto`, which corresponds to `JMSReplyTo`

Table 107–1 (Cont.) JMS Types—Member and Static Subprograms

Subprogram	Description
GET_TYPE Member Function	Returns `type`, which corresponds to `JMSType`
GET_USERID Member Function	Returns `userid`, which corresponds to `JMSXUserID`
GET_APPID Member Function	Returns `appid`, which corresponds to `JMSXAppID`
GET_GROUPID Member Function	Returns `groupid`, which corresponds to `JMSXGroupID`
GET_GROUPSEQ Member Function	Returns `groupseq`, which corresponds to `JMSXGroupSeq`
GET_BOOLEAN_ PROPERTY Member Function	Returns a BOOLEAN value if it can find `property_name` and if `java_type` is `BOOLEAN`
GET_BYTE_PROPERTY Member Function	Returns a `byte` value if it can find `property_name` and if `java_type` is `byte`
GET_SHORT_PROPERTY Member Function	Returns a `short` value if it can find `property_name` and if `java_type` is `short`
GET_INT_PROPERTY Member Function	Returns an `integer` value if it can find `property_name` and if `java_type` is `INT`
GET_LONG_PROPERTY Member Function	Returns a `number` value if it can find `property_name` and if `java_type` is `long`
GET_FLOAT_PROPERTY Member Function	Returns a `FLOAT` value if it can find `property_name` and if `java_type` is `FLOAT`
GET_DOUBLE_ PROPERTY Member Function	Returns a `DOUBLE PRECISION` value if it can find `property_name` and if `java_type` is `DOUBLE`
GET_STRING_PROPERTY Member Function	Returns a `VARCHAR` value if it can find `property_name` and if `java_type` is `STRING`
CONSTRUCT Static Function	Obtains instances of `aq$_jms_message`, which can hold a specific type of JMS message (`JMSText`, `JMSBytes`, `JMSMap`, `JMSStream`).
SET_TEXT Member Procedure	Sets the payload to an internal representation. See "Usage Notes" on page 107-28.

Table 107–1 (Cont.) JMS Types—Member and Static Subprograms

Subprogram	Description
GET_TEXT Member Procedure	Puts the internal representation of the payload into a VARCHAR2 or CLOB variable payload. See "Usage Notes" on page 107-29.
SET_BYTES Member Procedure	Sets the payload to an internal representation. See "Usage Notes" on page 107-30.
GET_BYTES Member Procedure	Puts the internal representation of the payload into a RAW or BLOB variable payload. See "Usage Notes" on page 107-31.

LOOKUP_PROPERTY_NAME Member Procedure

This procedure checks whether new_property_name exists in the properties.

Syntax

```
DBMS_AQJMS.LOOKUP_PROPERTY_NAME(
   new_property_name  IN  VARCHAR);
```

Parameters

Table 107–2 LOOKUP_PROPERTY_NAME Procedure Parameters

Parameter	Description
new_property_name	The property name to look up in the JMS property list

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

SET_REPLYTO Member Procedure

This procedure sets the replyto parameter, which corresponds to JMSReplyTo.

Syntax

```
DBMS_AQJMS.SET_REPLYTO(
   replyto IN SYS.AQ$_AGENT);
```

Parameters

Table 107–3 SET_REPLYTO Procedure Parameters

Parameter	Description
replyto	The client-supplied JMSReplyTo header field of the JMS message, which provides the destination for the reply to the message.

SET_TYPE Member Procedure

This procedure sets the JMS type, which can be any text, and which corresponds to JMSType.

Syntax

```
DBMS_AQJMS.SET_TYPE(
   type IN VARCHAR);
```

Parameters

Table 107–4 SET_TYPE Procedure Parameters

Parameter	Description
type	The JMSType header field of the JMS message, which is a client-supplied message type identifier

SET_USERID Member Procedure

This procedure sets userid, which corresponds to JMSXUserID.

Syntax

```
DBMS_AQJMS.SET_USERID(
   userid IN VARCHAR);
```

Parameters

Table 107–5 SET_USERID Procedure Parameters

Parameter	Description
userid	The JMS-defined JMSXUserID message property that is set by OJMS on send and contains the identity of the user sending the message

SET_APPID Member Procedure

This procedure sets appid, which corresponds to JMSXAppID.

Syntax

```
DBMS_AQJMS.SET_APPID(
   appid IN VARCHAR);
```

Parameters

Table 107–6 SET_APPID Procedure Parameters

Parameter	Description
appid	The JMS-defined JMSXAppID message property that is set by OJMS on send and contains the identity of the application sending the message

SET_GROUPID Member Procedure

This procedure sets groupid, which corresponds to JMSXGroupID.

Syntax

```
DBMS_AQJMS.SET_GROUPID(
   groupid IN VARCHAR);
```

Parameters

Table 107–7 SET_GROUPID Procedure Parameters

Parameter	Description
groupid	The JMS-defined JMSXGroupID message property that is set by the client and contains the identity of the message group of which the current message is a part

SET_GROUPSEQ Member Procedure

This procedure sets groupseq, which corresponds to JMSXGroupSeq.

Syntax

```
DBMS_AQJMS.SET_GROUPSEQ(
   groupseq IN INT);
```

Parameters

Table 107–8 SET_GROUPSEQ Procedure Parameters

Parameter	Description
groupseq	The JMS-defined JMSXGroupSeq message property that is set by the client and contains the sequence of the message within the group starting with 1.

CLEAR_PROPERTIES Member Procedure

This procedure clears all properties.

Syntax

```
DBMS_AQJMS.CLEAR_PROPERTIES;
```

SET_BOOLEAN_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure stores property_value in an internal representation (a NUMBER type).

Syntax

```
DBMS_AQJMS.SET_BOOLEAN_PROPERTY(
   property_name  IN VARCHAR,
```

```
   property_value IN BOOLEAN);
```

Parameters

Table 107–9 SET_BOOLEAN_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

SET_BYTE_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -127 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the byte datatype.

Syntax

```
DBMS_AQJMS.SET_BYTE_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);
```

Parameters

Table 107–10 SET_BYTE_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

ORA-24193 if the property value exceeds the valid range

SET_SHORT_PROPERTY Member Procedure

This procedure checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32767 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype.

Syntax

```
DBMS_AQJMS.SET_SHORT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);
```

Parameters

Table 107–11 SET_SHORT_PROPERTY Procedure Parameters

Parameter	Description
`property_name`	The name of the user-specified JMS user property or the JMS-specified JMS system property
`property_value`	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

ORA-24193 if the property value exceeds the valid range

SET_INT_PROPERTY Member Procedure

This procedure checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483647 to 2147483647

(32-bits). This check is necessary because in PL/SQL and the Oracle database, the INT datatype is 38 bits.

Syntax

```
DBMS_AQJMS.SET_INT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);
```

Parameters

Table 107–12 SET_INT_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

ORA-24193 if the property value exceeds the valid range

SET_LONG_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure stores property_value. In PL/SQL and Oracle database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed.

Syntax

```
DBMS_AQJMS.SET_LONG_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN NUMBER);
```

Parameters

Table 107–13 SET_LONG_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

SET_FLOAT_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure stores property_value.

Syntax

```
DBMS_AQJMS.SET_FLOAT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN FLOAT);
```

Parameters

Table 107–14 SET_FLOAT_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

SET_DOUBLE_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure stores property_value.

Syntax

```
DBMS_AQJMS.SET_DOUBLE_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN DOUBLE PRECISION);
```

Parameters

Table 107–15 SET_DOUBLE_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

SET_STRING_PROPERTY Member Procedure

This procedure checks whether property_name is null or exists. If not, the procedure stores property_value.

Syntax

```
DBMS_AQJMS.SET_STRING_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN VARDHAR);
```

Parameters

Table 107–16 SET_STRING_PROPERTY Procedure Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property
property_value	The value of the JMS message user property or system property

Exceptions

ORA-24191 if the property name exists

ORA-24192 if the property name is null

GET_REPLYTO Member Function

This function returns replyto, which corresponds to JMSReplyTo.

Syntax

```
DBMS_AQJMS.GET_REPLYTO(
   replyto OUT SYS.AQ$_AGENT);
```

Returns

Table 107–17 GET_REPLYTO Function Returns

Return	Description
replyto	The client-supplied JMSReplyTo header field of the JMS message, which provides the destination for the reply to the message.

GET_TYPE Member Function

This function returns type, which corresponds to JMSType.

Syntax

```
DBMS_AQJMS.GET_TYPE(
   type OUT VARCHAR);
```

Returns

Table 107–18 GET_TYPE Function Returns

Return	Description
type	The JMSType header field of the JMS message, which is a client-supplied message type identifier

GET_USERID Member Function

This function returns userid, which corresponds to JMSXUserID.

Syntax

```
DBMS_AQJMS.GET_USERID(
   userid OUT VARCHAR);
```

Returns

Table 107–19 GET_USERID Function Returns

Return	Description
userid	The JMS-defined JMSXUserID message property that is set by OJMS on send and contains the identity of the user sending the message

GET_APPID Member Function

This function returns appid, which corresponds to JMSXAppID.

Syntax

```
DBMS_AQJMS.GET_APPID(
   appid OUT VARCHAR);
```

Returns

Table 107–20 GET_APPID Function Returns

Return	Description
appid	The JMS-defined JMSXAppID message property that is set by OJMS on send and contains the identity of the application sending the message

GET_GROUPID Member Function

This function returns `groupid`, which corresponds to `JMSXGroupID`.

Syntax

```
DBMS_AQJMS.GET_GROUPID(
   groupid OUT VARCHAR);
```

Returns

Table 107–21 GET_GROUPID Function Returns

Return	Description
`groupid`	The JMS-defined `JMSXGroupID` message property that is set by the client and contains the identity of the message group of which the current message is a part

GET_GROUPSEQ Member Function

This function returns `groupseq`, which corresponds to `JMSXGroupSeq`.

Syntax

```
DBMS_AQJMS.GET_GROUPSEQ(
   groupseq OUT INT);
```

Returns

Table 107–22 GET_GROUPSEQ Function Returns

Return	Description
`groupseq`	The JMS-defined `JMSXGroupSeq` message property that is set by the client and contains the sequence of the message within the group starting with 1.

GET_BOOLEAN_PROPERTY Member Function

This function returns a BOOLEAN value if it can find `property_name` and if `java_type` is `BOOLEAN`. Otherwise it returns a `NULL`.

Syntax

```
DBMS_AQJMS.GET_BOOLEAN_PROPERTY(
```

```
  property_name IN  VARCHAR,
RETURN BOOLEAN);
```

Parameters

Table 107–23 GET_BOOLEAN_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_BYTE_PROPERTY Member Function

This function returns a byte value if it can find property_name and if java_type is byte. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_BYTE_PROPERTY(
   property_name IN  VARCHAR,
  RETURN  INT);
```

Parameters

Table 107–24 GET_BYTE_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_SHORT_PROPERTY Member Function

This function returns a short value if it can find property_name and if java_type is short. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_SHORT_PROPERTY(
   property_name IN  VARCHAR,
  RETURN INT);
```

Parameters

Table 107–25 GET_SHORT_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_INT_PROPERTY Member Function

This function returns an integer value if it can find property_name and if java_type is INT. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_INT_PROPERTY(
   property_name IN  VARCHAR,
  RETURN INT);
```

Parameters

Table 107–26 GET_INT_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_LONG_PROPERTY Member Function

This function returns a number value if it can find property_name and if java_type is long. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_LONG_PROPERTY(
   property_name IN  VARCHAR,
  RETURN NUMBER);
```

Parameters

Table 107–27 GET_LONG_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_FLOAT_PROPERTY Member Function

This function returns a FLOAT value if it can find property_name and if java_type is FLOAT. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_FLOAT_PROPERTY(
   property_name IN  VARCHAR,
  RETURN FLOAT);
```

Parameters

Table 107–28 GET_FLOAT_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_DOUBLE_PROPERTY Member Function

This function returns a DOUBLE PRECISION value if it can find property_name and if java_type is DOUBLE. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_DOUBLE_PROPERTY(
   property_name IN  VARCHAR,
  RETURN DOUBLE PRECISION);
```

Parameters

Table 107–29 GET_DOUBLE_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

GET_STRING_PROPERTY Member Function

This function returns a VARCHAR value if it can find property_name and if java_type is STRING. Otherwise it returns a NULL.

Syntax

```
DBMS_AQJMS.GET_STRING_PROPERTY(
   property_name IN  VARCHAR,
  RETURN VARCHAR);
```

Parameters

Table 107–30 GET_STRING_PROPERTY Parameters

Parameter	Description
property_name	The name of the user-specified JMS user property or the JMS-specified JMS system property

CONSTRUCT Static Function

This function is used to obtain instances of aq$_jms_message, which can hold a specific type of JMS message (JMSText, JMSBytes, JMSMap, JMSStream). The type of message each of these instances can hold depends on the mtype parameter passed to the contruct method. Once a message has been constructed, it can be used only to store JMS messages of the type it has been constructed to hold. The legal values of the mtype parameter are defined in the "Constants to Support the aq$_jms_message Type" on page 107-2. See "aq$_jms_message Type" on page 107-4 for more information.

Syntax

```
DBMS_AQJMS.CONSTRUCT(
   mtype  IN INT)
  RETURN aq$_jms_message;
```

Syntax

```
DBMS_AQJMS.CONSTRUCT RETURN aq$_jms_text_message;
```

SET_TEXT Member Procedure

This procedure sets the payload, a VARCHAR2, to an internal representation. If the payload length is <= 4000, it is set into text_vc. Otherwise, it is set into text_lob.

Syntax

```
DBMS_AQJMS.SET_TEXT(
   payload IN VARCHAR2;
```

Syntax

This procedure sets the payload, a CLOB, to an internal representation (sets payload into text_lob).

```
DBMS_AQJMS.SET_TEXT(
   payload IN CLOB;
```

Parameters

Table 107–31 SET_TEXT Procedure Parameters

Parameter	Description
payload	The payload of a JMS message

Usage Notes

This procedure is available with aq$_jms_text_message (and aq$_jms_message), but not aq$_jms_bytes_message.

GET_TEXT Member Procedure

This procedure puts the internal representation of the payload into a VARCHAR2 variable payload. It puts text_vc into payload if text_vc is not null, or transfers text_lob into payload if the length of text_lob is =< 32767 (2**16 -1).

Syntax

```
DBMS_AQJMS.GET_TEXT(
```

```
   payload OUT VARCHAR2);
```

Syntax

This procedure puts the internal payload into a CLOB variable payload. It puts text_lob into payload if text_lob is not null, or transfers text_vc into payload.

```
DBMS_AQJMS.GET_TEXT(
   payload OUT CLOB;
```

Parameters

Table 107–32 GET_TEXT Procedure Parameters

Parameter	Description
payload	The payload of a JMS message

Exceptions

ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of VARCHAR2 in PL/SQL).

Usage Notes

This procedure is available with aq$_jms_text_message (and aq$_jms_message), but not aq$_jms_bytes_message.

SET_BYTES Member Procedure

This procedure sets the payload, a RAW value, to an internal representation (into bytes_raw if the length of payload is <= 2000; otherwise into bytes_lob).

Syntax

```
DBMS_AQJMS.SET_BYTES(
   payload IN RAW);
```

Syntax

This procedure sets the payload, a BLOB value, to an internal representation (into bytes_lob).

```
DBMS_AQJMS.SET_BYTES(
```

```
   payload IN BLOB);
```

Parameters

Table 107–33 SET_BYTES Procedure Parameters

Parameter	Description
payload	The payload of a JMS message

Usage Notes

This procedure is available with aq$_jms_bytes_message (and aq$_jms_message), but not aq$_jms_text_message.

GET_BYTES Member Procedure

This procedure puts the internal representation of the payload into a RAW variable payload. It puts bytes_raw into payload if it is not null, or transfers bytes_lob into payload if the length of bytes_lob is =< 32767 (2**16 -1).

Syntax

```
DBMS_AQJMS.GET_BYTES(
   payload OUT RAW);
```

Syntax

This procedure puts the internal representation of the payload into a BLOB variable payload.

```
DBMS_AQJMS.GET_BYTES(
   payload OUT BLOB);
```

Exceptions

ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of VARCHAR2 in PL/SQL).

Returns

Table 107–34 GET_BYTES Function Returns

Return	Description
payload	The payload of a JMS message

Usage Notes

This procedure is available with aq$_jms_bytes_message (and aq$_jms_message), but not aq$_jms_text_message.

Enqueuing Through the Oracle JMS Administrative Interface: Example

The following sample program enqueues a large text message (along with JMS user properties) in an AQ queue created through the OJMS administrative interfaces to hold JMS TEXT messages. Both the text and bytes messages enqueued in this example can be dequeued using OJMS Java clients.

```
DECLARE

    text        varchar2(32767);
    agent       sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
    message     sys.aq$_jms_text_message;

    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid               raw(16);

BEGIN

    message := sys.aq$_jms_text_message.construct;

    message.set_replyto(agent);
    message.set_type('tkaqpet2');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    message.set_boolean_property('import', True);
    message.set_string_property('color', 'RED');
    message.set_short_property('year', 1999);
    message.set_long_property('mileage', 300000);
```

```
    message.set_double_property('price', 16999.99);
    message.set_byte_property('password', 127);

    FOR i IN 1..500 LOOP
        text := CONCAT (text, '1234567890');
    END LOOP;

    message.set_text(text);

    dbms_aq.enqueue(queue_name => 'jmsuser.jms_text_t1',
                      enqueue_options => enqueue_options,
                      message_properties => message_properties,
                      payload => message,
                      msgid => msgid);

END;
```

The following sample program enqueues a large bytes message.

```
DECLARE

    text        VARCHAR2(32767);
    bytes       RAW(32767);
    agent       sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
    message     sys.aq$_jms_bytes_message;
    body        BLOB;
    position    INT;

    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);

BEGIN

    message := sys.aq$_jms_bytes_message.construct;

    message.set_replyto(agent);
    message.set_type('tkaqper4');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq_raw');
    message.set_groupid('st');
    message.set_groupseq(1);

    message.set_boolean_property('import', True);
    message.set_string_property('color', 'RED');
```

```
    message.set_short_property('year', 1999);
    message.set_long_property('mileage', 300000);
    message.set_double_property('price', 16999.99);

--  prepare a huge payload into a blob

    FOR i IN 1..1000 LOOP
        text := CONCAT (text, '0123456789ABCDEF');
    END LOOP;

    bytes := HEXTORAW(text);

    dbms_lob.createtemporary(lob_loc => body, cache => TRUE);
    dbms_lob.open (body, DBMS_LOB.LOB_READWRITE);
    position := 1 ;
    FOR i IN 1..10 LOOP
        dbms_lob.write ( lob_loc => body,
                amount => FLOOR((LENGTH(bytes)+1)/2),
                offset => position,
                buffer => bytes);
        position := position + FLOOR((LENGTH(bytes)+1)/2) ;
    END LOOP;

-- end of the preparation

    message.set_bytes(body);
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_bytes_t1',
                      enqueue_options => enqueue_options,
                      message_properties => message_properties,
                      payload => message,
                      msgid => msgid);

    dbms_lob.freetemporary(lob_loc => body);
END;
```

108

Logical Change Record Types

This chapter describes the logical change record (LCR) types. In Streams, LCRs are message payloads that contain information about changes to a database. These changes can include changes to the data, which are data manipulation language (DML) changes, and changes to database objects, which are data definition language (DDL) changes.

When you use Streams, the capture process captures changes in the form of LCRs and enqueues them into a queue. These LCRs can be propagated from a queue in one database to a queue in another database. Finally, the apply process can apply LCRs at a destination database. You also have the option of creating, enqueuing, and dequeuing LCRs manually.

LCR types are used with the following Oracle-supplied PL/SQL packages:

- DBMS_APPLY_ADM
- DBMS_AQ
- DBMS_AQADM
- DBMS_CAPTURE_ADM
- DBMS_PROPAGATION_ADM
- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_TRANSFORM

This chapter contains these topics:

- LCR$_DDL_RECORD Type
- LCR$_ROW_RECORD Type
- Common Subprograms for LCR$_ROW_RECORD and LCR$_DDL_RECORD
- LCR$_ROW_LIST Type
- LCR$_ROW_UNIT Type

LCR$_DDL_RECORD Type

This type represents a DDL change to a database object.

If you create or modify a DDL LCR, then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

LCR$_DDL_RECORD Constructor

Creates a SYS.LCR$_DDL_RECORD object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name       IN VARCHAR2,
   command_type               IN VARCHAR2,
   object_owner               IN VARCHAR2,
   object_name                IN VARCHAR2,
   object_type                IN VARCHAR2,
   ddl_text                   IN CLOB,
   logon_user                 IN VARCHAR2,
   current_schema             IN VARCHAR2,
   base_table_owner           IN VARCHAR2,
   base_table_name            IN VARCHAR2,
   tag                        IN RAW       DEFAULT NULL,
   transaction_id             IN VARCHAR2  DEFAULT NULL,
   scn                        IN NUMBER    DEFAULT NULL)
RETURN SYS.LCR$_DDL_RECORD;
```

LCR$_DDL_RECORD Constructor Function Parameters

Table 108–1 Constructor Function Parameters for LCR$_DDL_RECORD **(Page 1 of 3)**

Parameter	Description
source_database_name	The database where the DDL statement occurred. If you do not include the domain name, then the local domain is appended to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then DBS1.NET is specified automatically. This parameter should be set to a non-NULL value.
command_type	The type of command executed in the DDL statement. This parameter should be set to a non-NULL value. **See Also:** The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types The following command types *are not supported* in DDL LCRs: ALTER MATERIALIZED VIEW ALTER MATERIALIZED VIEW LOG ALTER SUMMARY CREATE SCHEMA CREATE MATERIALIZED VIEW CREATE MATERIALIZED VIEW LOG CREATE SUMMARY DROP MATERIALIZED VIEW DROP MATERIALIZED VIEW LOG DROP SUMMARY RENAME The snapshot equivalents of the materialized view command types are also not supported.
object_owner	The user who owns the object on which the DDL statement was executed
object_name	The database object on which the DDL statement was executed

Table 108–1 Constructor Function Parameters for LCR$_DDL_RECORD **(Page 2 of 3)**

Parameter	Description
object_type	The type of object on which the DDL statement was executed. The following are valid object types: CLUSTER FUNCTION INDEX LINK OUTLINE PACKAGE PACKAGE BODY PROCEDURE SEQUENCE SYNONYM TABLE TRIGGER TYPE USER VIEW LINK represents a database link. NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.
ddl_text	The text of the DDL statement. This parameter should be set to a non-NULL value.
logon_user	The user whose session executed the DDL statement
current_schema	The schema that is used if no schema is specified explicitly for the modified database objects in ddl_text. If a schema is specified in ddl_text that differs from the one specified for current_schema, then the schema specified in ddl_text is used. This parameter should be set to a non-NULL value.
base_table_owner	If the DDL statement is a table related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_owner specifies the owner of the table involved. Otherwise, base_table_owner is NULL.

Table 108–1 Constructor Function Parameters for LCR$_DDL_RECORD **(Page 3 of 3)**

Parameter	Description
base_table_name	If the DDL statement is a table related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_name specifies the name of the table involved. Otherwise, base_table_name is NULL.
tag	A binary tag that enables tracking of the LCR. For example, this tag may be used to determine the original source database of the DDL statement if apply forwarding is used. **See Also:** *Oracle9i Streams* for more information about tags
transaction_id	The identifier of the transaction
scn	The SCN at the time when the change record for a captured LCR was written to the redo. The SCN value is meaningless for a user-created LCR.

Summary of LCR$_DDL_RECORD Subprograms

Table 108–2 LCR$_DDL_RECORD Subprograms

Subprogram	Description
Common Subprograms	See "Common Subprograms for LCR$_ROW_RECORD and LCR$_DDL_RECORD" on page 108-33 for a list of subprograms common to the SYS.LCR$_ROW_RECORD and SYS.LCR$_DDL_RECORD types
"EXECUTE Member Procedure" on page 108-9	Executes the LCR under the security domain of the current user
"GET_BASE_TABLE_NAME Member Function" on page 108-9	Returns the base (dependent) table name
"GET_BASE_TABLE_OWNER Member Function" on page 108-9	Returns the base (dependent) table owner
"GET_CURRENT_SCHEMA Member Function" on page 108-9	Returns the default schema (user) name
"GET_DDL_TEXT Member Procedure" on page 108-10	Gets the DDL text in a CLOB
"GET_LOGON_USER Member Function" on page 108-11	Returns the logon user name
"GET_OBJECT_TYPE Member Function" on page 108-11	Returns the type of the object involved for the DDL
"SET_BASE_TABLE_NAME Member Procedure" on page 108-11	Sets the base (dependent) table name
"SET_BASE_TABLE_OWNER Member Procedure" on page 108-12	Sets the base (dependent) table owner
"SET_CURRENT_SCHEMA Member Procedure" on page 108-12	Sets the default schema (user) name
"SET_DDL_TEXT Member Procedure" on page 108-13	Sets the DDL text
"SET_LOGON_USER Member Procedure" on page 108-13	Sets the logon user name
"SET_OBJECT_TYPE Member Procedure" on page 108-14	Sets the object type

EXECUTE Member Procedure

Executes the DDL LCR under the security domain of the current user. Any apply process handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

> **Note:** The EXECUTE member procedure can be invoked only in an apply handler for an apply process.

Syntax

```
MEMBER PROCEDURE EXECUTE();
```

GET_BASE_TABLE_NAME Member Function

Returns the base (dependent) table name.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_NAME RETURN VARCHAR2;
```

GET_BASE_TABLE_OWNER Member Function

Returns the base (dependent) table owner.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_OWNER RETURN VARCHAR2;
```

GET_CURRENT_SCHEMA Member Function

Returns the current schema name.

Syntax

```
MEMBER FUNCTION GET_CURRENT_SCHEMA RETURN VARCHAR2;
```

GET_DDL_TEXT Member Procedure

Gets the DDL text in a CLOB.

The following is an example of a PL/SQL procedure that uses this procedure to get the DDL text in a DDL LCR:

```
CREATE OR REPLACE PROCEDURE ddl_in_lcr (ddl_lcr in SYS.LCR$_DDL_RECORD)
IS
  ddl_text   CLOB;
BEGIN
  DBMS_OUTPUT.PUT_LINE( '  ----------------------------------------' );
  DBMS_OUTPUT.PUT_LINE( '  Displaying DDL text in a DDL LCR: ' );
  DBMS_OUTPUT.PUT_LINE( '  ----------------------------------------' );
  DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
  ddl_lcr.GET_DDL_TEXT(ddl_text);
  DBMS_OUTPUT.PUT_LINE('DDL text:' || ddl_text);
  DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/
```

Note: GET_DDL_TEXT is a member procedure and not a member function to make it easier for you to manage the space used by the CLOB. Notice that the previous example creates temporary space for the CLOB and then frees the temporary space when it is no longer needed.

Syntax

```
MEMBER FUNCTION GET_DDL_TEXT
  ddl_text  IN OUT  CLOB);
```

Parameter

Table 108–3 GET_DDL_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text in the DDL LCR

GET_LOGON_USER Member Function

Returns the logon user name.

Syntax

```
MEMBER FUNCTION GET_LOGON_USER RETURN VARCHAR2;
```

GET_OBJECT_TYPE Member Function

Returns the type of the object involved for the DDL.

Syntax

```
MEMBER FUNCTION GET_OBJECT_TYPE RETURN VARCHAR2;
```

SET_BASE_TABLE_NAME Member Procedure

Sets the base (dependent) table name.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_NAME(
     base_table_name IN VARCHAR2);
```

Parameter

Table 108–4 SET_BASE_TABLE_NAME Procedure Parameter

Parameter	Description
base_table_name	The name of the base table

SET_BASE_TABLE_OWNER Member Procedure

Sets the base (dependent) table owner.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_OWNER(
     base_table_owner IN VARCHAR2);
```

Parameter

Table 108–5 SET_BASE_TABLE_OWNER Procedure Parameter

Parameter	Description
base_table_owner	The name of the base owner

SET_CURRENT_SCHEMA Member Procedure

Sets the default schema (user) name.

Syntax

```
MEMBER PROCEDURE SET_CURRENT_SCHEMA(
     current_schema IN VARCHAR2);
```

Parameter

Table 108–6 SET_CURRENT_SCHEMA Procedure Parameter

Parameter	Description
current_schema	The name of the schema to set as the current schema. This parameter should be set to a non-NULL value.

SET_DDL_TEXT Member Procedure

Sets the DDL text.

Syntax

```
MEMBER PROCEDURE SET_DDL_TEXT(
    ddl_text     IN CLOB);
```

Parameter

Table 108–7 SET_DDL_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text. This parameter should be set to a non-NULL value.

SET_LOGON_USER Member Procedure

Sets the logon user name.

Syntax

```
MEMBER PROCEDURE SET_LOGON_USER(
     logon_user IN VARCHAR2);
```

Parameter

Table 108–8 SET_LOGON_USER Procedure Parameter

Parameter	Description
logon_user	The name of the schema to set as the logon user

SET_OBJECT_TYPE Member Procedure

Sets the object type.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_TYPE(
     object_type    IN VARCHAR2);
```

Parameter

Table 108–9 SET_OBJECT_TYPE Procedure Parameter

Parameter	Description
object_type	The object type. The following are valid object types: CLUSTER FUNCTION INDEX LINK OUTLINE PACKAGE PACKAGE BODY PROCEDURE SEQUENCE SYNONYM TABLE TRIGGER TYPE USER VIEW LINK represents a database link. NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.

LCR$_ROW_RECORD Type

This type represents a DML change to a row in a table. This type uses the `LCR$_ROW_LIST` type.

If you create or modify a row LCR, then make sure the `command_type` attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

See Also: "LCR$_ROW_LIST Type" on page 108-40

LCR$_ROW_RECORD Constructor

Creates a SYS.LCR$_ROW_RECORD object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name       IN VARCHAR2,
   command_type               IN VARCHAR2,
   object_owner               IN VARCHAR2,
   object_name                IN VARCHAR2,
   tag                        IN RAW                DEFAULT NULL,
   transaction_id             IN VARCHAR2           DEFAULT NULL,
   scn                        IN NUMBER             DEFAULT NULL,
   old_values                 IN SYS.LCR$_ROW_LIST  DEFAULT NULL,
   new_values                 IN SYS.LCR$_ROW_LIST  DEFAULT NULL)
RETURN SYS.LCR$_ROW_RECORD;
```

LCR$_ROW_RECORD Constructor Function Parameters

Table 108–10 Constructor Function Parameters for LCR$_ROW_RECORD **(Page 1 of 2)**

Parameter	Description
source_database_name	The database where the row change occurred. If you do not include the domain name, then the local domain is appended to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then DBS1.NET is specified automatically. This parameter should be set to a non-NULL value.
command_type	The type of command executed in the DML statement. This parameter should be set to a non-NULL value. Valid values are the following: INSERT UPDATE DELETE LOB ERASE LOB WRITE LOB TRIM If INSERT, then an LCR should have a new_values collection that is not empty and an empty or NULL old_values collection. If UPDATE, then an LCR should have a new_values collection that is not empty and an old_values collection that is not empty. If DELETE, then an LCR should have a NULL or empty new_values collection and an old_values collection that is not empty. If LOB ERASE, LOB WRITE, or LOB TRIM, then an LCR should have a new_values collection that is not empty and an empty or NULL old_values collection.
object_owner	The user who owns the table on which the row change occurred. This parameter should be set to a non-NULL value.
object_name	The table on which the DML statement was executed. This parameter should be set to a non-NULL value.
tag	A binary tag that enables tracking of the LCR. For example, this tag may be used to determine the original source database of the DML change when apply forwarding is used. **See Also:** *Oracle9i Streams* for more information about tags
transaction_id	The identifier of the transaction

Table 108–10 Constructor Function Parameters for LCR$_ROW_RECORD **(Page 2 of 2)**

Parameter	Description
scn	The SCN at the time when the change record was written to the redo
old_values	If the DML statement is an UPDATE or a DELETE statement, then the values of columns in the row before the DML statement
new_values	If the DML statement is an UPDATE or an INSERT statement, then the values of columns in the row after the DML statement. If the LCR reflects a LOB operation, then the supplementally logged columns and any relevant LOB information.

Summary of LCR$_ROW_RECORD Subprograms

Table 108–11 LCR$_ROW_RECORD Subprograms **(Page 1 of 2)**

Subprogram	Description
Common Subprograms	See "Common Subprograms for LCR$_ROW_RECORD and LCR$_DDL_RECORD" on page 108-33 for a list of subprograms common to the SYS.LCR$_ROW_RECORD and SYS.LCR$_DDL_RECORD types
"ADD_COLUMN Member Procedure" on page 108-20	Adds the value as old or new, depending on the value type specified, for the column
"DELETE_COLUMN Member Procedure" on page 108-21	Deletes the old value, the new value, or both, for the specified column, depending on the value type specified
"EXECUTE Member Procedure" on page 108-22	Executes the LCR under the security domain of the current user
"GET_LOB_INFORMATION Member Function" on page 108-23	Gets the LOB information for the column
"GET_LOB_OFFSET Member Function" on page 108-24	Returns the LOB offset for the specified column
"GET_LOB_OPERATION_SIZE Member Function" on page 108-25	Gets the operation size for the LOB column
"GET_VALUE Member Function" on page 108-26	Returns the old or new value for the specified column, depending on the value type specified
"GET_VALUES Member Function" on page 108-26	Returns a list of old or new values, depending on the value type specified
"RENAME_COLUMN Member Procedure" on page 108-27	Renames a column in an LCR
"SET_LOB_INFORMATION Member Procedure" on page 108-28	Sets LOB information for the column
"SET_LOB_OFFSET Member Procedure" on page 108-29	Sets the LOB offset for the specified column

Table 108–11 LCR$_ROW_RECORD Subprograms **(Page 2 of 2)**

Subprogram	Description
"SET_LOB_OPERATION_SIZE Member Procedure" on page 108-30	Sets the operation size for the LOB column
"SET_VALUE Member Procedure" on page 108-31	Overwrites the value of the specified column
"SET_VALUES Member Procedure" on page 108-32	Replaces the existing old or new values for the LCR, depending on the value type specified

ADD_COLUMN Member Procedure

Adds the value as old or new, depending on the value type specified, for the column. An error is raised if a value of the same type already exists for the column.

To set a column value that already exists, run SET_VALUE.

See Also: "SET_VALUE Member Procedure" on page 108-31

Syntax

```
MEMBER PROCEDURE ADD_COLUMN(
   value_type     IN VARCHAR2,
   column_name    IN VARCHAR2,
   column_value   IN SYS.AnyData);
```

Parameters

Table 108–12 ADD_COLUMN Procedure Parameters

Parameter	Description
value_type	The type of value to add for the column. Specify old to add the old value of the column. Specify new to add the new value of the column.
column_name	The column name. This name is not validated. An error may be raised during application of the LCRs if an invalid name is specified.
column_value	The value of the column. If NULL, then an error is raised. A NULL column value can be specified by encapsulating the NULL value in a SYS.AnyData wrapper.

DELETE_COLUMN Member Procedure

Deletes the old value, the new value, or both, for the specified column, depending on the value type specified.

Syntax

```
MEMBER PROCEDURE DELETE_COLUMN(
   column_name    IN VARCHAR2,
   value_type     IN VARCHAR2  DEFAULT '*');
```

Parameters

Table 108–13 DELETE_COLUMN Procedure Parameters

Parameter	Description
`column_name`	The column name. An error is raised if the column does not exist in the LCR.
`value_type`	The type of value to delete for the column. Specify `old` to delete the old value of the column. Specify `new` to delete the new value of the column. If * is specified, then both the old and new values are deleted.

EXECUTE Member Procedure

Executes the row LCR under the security domain of the current user. Any apply process handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

> **Note:** The EXECUTE member procedure can be invoked only in an apply handler for an apply process.

Syntax

```
MEMBER PROCEDURE EXECUTE(
   conflict_resolution    IN BOOLEAN);
```

Parameters

Table 108–14 EXECUTE Procedure Parameters

Parameter	Description
conflict_resolution	If true, then any conflict resolution defined for the table using the SET_UPDATE_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package is used to resolve conflicts resulting from the execution of the LCR. If false, then conflict resolution is not used.

GET_LOB_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;
DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;
DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;
DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;
```

Returns `NULL` if the specified column does not exist.

Syntax

```
MEMBER FUNCTION GET_LOB_INFORMATION(
  value_type   IN VARCHAR2,
  column_name  IN VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 108–15 GET_LOB_INFORMATION Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column, either `old` or `new`
`column_name`	The name of the column

GET_LOB_OFFSET Member Function

Gets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB. That is, the information is DBMS_LCR.LAST_LOB_CHUNK or DBMS_LCR.LOB_CHUNK
- The command type is LOB ERASE or LOB WRITE

Otherwise, returns NULL.

Syntax

```
GET_LOB_OFFSET(
   value_type      IN VARCHAR2,
   column_name     IN VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 108–16 GET_LOB_OFFSET Procedure Parameters

Parameter	Description
value_type	The type of value to return for the column. Currently, only new can be specified.
column_name	The name of the LOB column

GET_LOB_OPERATION_SIZE Member Function

Gets the operation size for the LOB column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB
- The command type is LOB ERASE or LOB TRIM
- The information is DBMS_LCR.LAST_LOB_CHUNK

Otherwise, returns NULL.

Syntax

```
MEMBER FUNCTION GET_LOB_OPERATION_SIZE(
  value_type    IN VARCHAR2,
  column_name   IN VARCHAR2)
RETURN NUMBER,
```

Parameters

Table 108–17 GET_LOB_OPERATION_SIZE Function Parameters

Parameter	Description
value_type	The type of value to return for the column. Currently, only new can be specified.
column_name	The name of the LOB column

GET_VALUE Member Function

Returns the old or new value for the specified column, depending on the value type specified.

Syntax

```
MEMBER FUNCTION GET_VALUE(
   value_type          IN VARCHAR2,
   column_name         IN VARCHAR2)
RETURN SYS.AnyData;
```

Parameters

Table 108–18 GET_VALUE Procedure Parameters

Parameter	Description
value_type	The type of value to return for the column. Specify old to get the old value for the column. Specify new to get the new value for the column.
column_name	The column name. If the column is present and has a NULL value, returns a SYS.AnyData instance containing a NULL value. If the column value is absent, returns a NULL.

GET_VALUES Member Function

Returns a list of old or new values, depending on the value type specified.

Syntax

```
MEMBER FUNCTION GET_VALUES(
   value_type    IN VARCHAR2)
RETURN SYS.LCR$_ROW_LIST;
```

Parameter

Table 108–19 GET_VALUES Procedure Parameter

Parameter	Description
value_type	The type of values to return. Specify old to return a list of old values. Specify new to return a list of new values.

RENAME_COLUMN Member Procedure

Renames a column in an LCR.

Syntax

```
MEMBER PROCEDURE RENAME_COLUMN(
   from_column_name   IN VARCHAR2,
   to_column_name     IN VARCHAR2,
   value_type         IN VARCHAR2  DEFAULT '*');
```

Parameters

Table 108–20 RENAME_COLUMN Procedure Parameters

Parameter	Description
`from_column_name`	The existing column name
`to_column_name`	The new column name. An error is raised if a column with the specified name already exists.
`value_type`	The type of value for which to rename the column. Specify `old` to rename the old value of the column. An error is raised if the old value does not exist in the LCR. Specify `new` to rename the new value of the column. An error is raised if the new value does not exist in the LCR. If * is specified, then the column names for both old and new value are renamed. An error is raised if either column value does not exist in the LCR.

SET_LOB_INFORMATION Member Procedure

Sets LOB information for the column.

Syntax

```
MEMBER PROCEDURE SET_LOB_INFORMATION(
  value_type          IN      VARCHAR2,
  column_name         IN      VARCHAR2,
  lob_information     IN      NUMBER);
```

Parameters

Table 108–21 SET_LOB_INFORMATION Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column, either old or new. Specify old only if lob_information is set to DBMS_LCR.NOT_A_LOB.
column_name	The name of the column. An exception is raised if the column value does not exist. You may need to set this parameter for non-LOB columns.
lob_information	Specify one of the following values: DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1; DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2; DBMS_LCR.INLINE_LOB CONSTANT NUMBER := 3; DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4; DBMS_LCR.LOB_CHUNK CONSTANT NUMBER := 5; DBMS_LCR.LAST_LOB_CHUNK CONSTANT NUMBER := 6;

SET_LOB_OFFSET Member Procedure

Sets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns.

Syntax

```
SET_LOB_OFFSET(
   value_type      IN VARCHAR2,
   column_name     IN VARCHAR2,
   lob_offset      IN NUMBER);
```

Parameters

Table 108–22 SET_LOB_OFFSET Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column. Currently, only new can be specified.
column_name	The column name. An error is raised if the column value does not exist in the LCR.
lob_offset	The LOB offset number. Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.

SET_LOB_OPERATION_SIZE Member Procedure

Sets the operation size for the LOB column in the number of characters for CLOB columns and bytes for BLOB columns.

Syntax

```
MEMBER PROCEDURE SET_LOB_OPERATION_SIZE(
  value_type          IN      VARCHAR2,
  column_name         IN      VARCHAR2,
  lob_operation_size  IN      NUMBER);
```

Parameters

Table 108–23 SET_LOB_OPERATION_SIZE Procedure Parameters

Parameter	Description
value_type	The type of value to set for the column. Currently, only new can be specified.
column_name	The name of the LOB column. An exception is raised if the column value does not exist in the LCR.
lob_operation_size	If lob_information for the LOB is or will be DBMS_LCR.LAST_LOB_CHUNK, then can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB_ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB_TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. Otherwise, set to NULL.

SET_VALUE Member Procedure

Overwrites the old or new value of the specified column.

One reason you may want to overwrite an old value for a column is to resolve an error that resulted from a conflict.

Syntax

```
MEMBER PROCEDURE SET_VALUE(
   value_type       IN VARCHAR2,
   column_name      IN VARCHAR2,
   column_value     IN SYS.AnyData);
```

Parameters

Table 108–24 SET_VALUE Procedure Parameters

Parameter	Description
`value_type`	The type of value to set. Specify `old` to set the old value of the column. Specify `new` to set the new value of the column.
`column_name`	The column name. An error is raised if the specified `column_value` does not exist in the LCR for the specified `column_type`.
`column_value`	The new value of the column. If `NULL` is specified, then an error is raised. To set the value to `NULL`, encapsulate the `NULL` in a `SYS.AnyData` instance.

SET_VALUES Member Procedure

Replaces all old values or all new values for the LCR, depending on the value type specified.

Syntax

```
MEMBER PROCEDURE SET_VALUES(
   value_type     IN VARCHAR2,
   value_list     IN SYS.LCR$_ROW_LIST);
```

Parameters

Table 108–25 SET_VALUES Procedure Parameters

Parameter	Description
value_type	The type of values to replace. Specify old to replace the old values. Specify new to replace the new values.
value_list	List of values to replace the existing list. Use a NULL or an empty list to remove all values.

Common Subprograms for LCR$_ROW_RECORD and LCR$_DDL_RECORD

The following functions and procedures are common to both the LCR$_ROW_RECORD and LCR$_DDL_RECORD type.

See Also: For descriptions of the subprograms for these types that are exclusive to each type:

- "Summary of LCR$_DDL_RECORD Subprograms" on page 108-8
- "Summary of LCR$_ROW_RECORD Subprograms" on page 108-19

Table 108–26 Summary of Common Subprograms for Row and DDL Types

Subprogram	Description
"GET_COMMAND_TYPE Member Function" on page 108-35	Returns the command type of the LCR
"GET_OBJECT_NAME Member Function" on page 108-35	Returns the name of the object that is changed by the LCR
"GET_OBJECT_OWNER Member Function" on page 108-35	Returns the owner of the object that is changed by the LCR
"GET_SCN Member Function" on page 108-35	Returns the system change number (SCN) of the LCR
"GET_SOURCE_DATABASE_NAME Member Function" on page 108-36	Returns the source database name.
"GET_TAG Member Function" on page 108-36	Returns the tag for the LCR
"GET_TRANSACTION_ID Member Function" on page 108-36	Returns the transaction identifier of the LCR
"IS_NULL_TAG Member Function" on page 108-36	Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`
"SET_COMMAND_TYPE Member Procedure" on page 108-37	Sets the command type
"SET_OBJECT_NAME Member Procedure" on page 108-38	Sets the name of the object that is changed by the LCR
"SET_OBJECT_OWNER Member Procedure" on page 108-38	Sets the owner of the object that is changed by the LCR
"SET_SOURCE_DATABASE_NAME Member Procedure" on page 108-39	Sets the source database name of the object that is changed by the LCR
"SET_TAG Member Procedure" on page 108-39	Sets the tag for the LCR

GET_COMMAND_TYPE Member Function

Returns the command type of the LCR.

> **See Also:** The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER FUNCTION GET_COMMAND_TYPE RETURN VARCHAR2;
```

GET_OBJECT_NAME Member Function

Returns the name of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_NAME RETURN VARCHAR2;
```

GET_OBJECT_OWNER Member Function

Returns the owner of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_OWNER RETURN VARCHAR2;
```

GET_SCN Member Function

Returns the system change number (SCN) of the LCR.

Syntax

```
MEMBER FUNCTION GET_SCN RETURN NUMBER;
```

GET_SOURCE_DATABASE_NAME Member Function

Returns the global name of the source database name. The source database is the database where the change occurred.

Syntax

```
MEMBER FUNCTION GET_SOURCE_DATABASE_NAME RETURN VARCHAR2;
```

GET_TAG Member Function

Returns the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag may be used to determine the original source database of the DML or DDL change when apply forwarding is used.

See Also: *Oracle9i Streams* for more information about tags

Syntax

```
MEMBER FUNCTION GET_TAG RETURN RAW;
```

GET_TRANSACTION_ID Member Function

Returns the transaction identifier of the LCR.

Syntax

```
MEMBER FUNCTION GET_TRANSACTION_ID RETURN VARCHAR2;
```

IS_NULL_TAG Member Function

Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`.

See Also: *Oracle9i Streams* for more information about tags

Syntax

```
MEMBER FUNCTION IS_NULL_TAG RETURN VARCHAR2;
```

SET_COMMAND_TYPE Member Procedure

Sets the command type. If the command type specified cannot be interpreted, then an error is raised. For example, changing `INSERT` to `GRANT` would raise an error.

See Also:

- The description of the `command_type` parameter in "LCR$_DDL_RECORD Constructor Function Parameters" on page 108-5
- The description of the command_type parameter in "LCR$_ROW_RECORD Constructor Function Parameters" on page 108-17
- The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER PROCEDURE SET_COMMAND_TYPE(
   command_type    IN VARCHAR2);
```

Parameter

Table 108–27 SET_COMMAND_TYPE Procedure Parameter

Parameter	Description
`command_type`	The command type. This parameter should be set to a non-`NULL` value.

SET_OBJECT_NAME Member Procedure

Sets the name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_NAME(
   object_name     IN VARCHAR2);
```

Parameter

Table 108–28 SET_OBJECT_NAME Procedure Parameter

Parameter	Description
object_name	The name of the object

SET_OBJECT_OWNER Member Procedure

Sets the owner of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_OWNER(
   object_owner IN VARCHAR2);
```

Parameter

Table 108–29 SET_OBJECT_OWNER Procedure Parameter

Parameter	Description
object_owner	The schema that contains the object

SET_SOURCE_DATABASE_NAME Member Procedure

Sets the source database name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_SOURCE_DATABASE_NAME(
   source_database_name    IN VARCHAR2);
```

Parameter

Table 108–30 SET_SOURCE_DATABASE_NAME Procedure Parameter

Parameter	Description
source_database_name	The source database of the change. If you do not include the domain name, then the local domain is appended to the database name automatically. For example, if you specify DBS1 and the local domain is .NET, then DBS1.NET is specified automatically. This parameter should be set to a non-NULL value.

SET_TAG Member Procedure

Sets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag may be used to determine the original source database of the change when apply forwarding is used.

See Also: *Oracle9i Streams* for more information about tags

Syntax

```
MEMBER PROCEDURE SET_TAG(
   tag    IN RAW);
```

Parameter

Table 108–31 SET_TAG Procedure Parameter

Parameter	Description
tag	The binary tag for the LCR. The size limit for a tag value is two kilobytes.

LCR$_ROW_LIST Type

Identifies a list of column values for a row in a table.

This type uses the LCR$_ROW_UNIT type and is used in the LCR$_ROW_RECORD type.

See Also:

- "LCR$_ROW_UNIT Type" on page 108-41
- "LCR$_ROW_RECORD Type" on page 108-15

Syntax

```
CREATE TYPE SYS.LCR$_ROW_LIST AS TABLE OF SYS.LCR$_ROW_UNIT
/
```

LCR$_ROW_UNIT Type

Identifies the value for a column in a row.

This type is used in the `LCR$_ROW_LIST` type.

See Also: "LCR$_ROW_LIST Type" on page 108-40

Syntax

```
CREATE TYPE LCR$_ROW_UNIT AS OBJECT (
  column_name         VARCHAR2(4000),
  data                SYS.AnyData,
  lob_information     NUMBER,
  lob_offset          NUMBER,
  lob_operation_size  NUMBER);
/
```

Attributes

Table 108–32 LCR$_ROW_UNIT Attributes

Attribute	Description
column_name	The name of the column
data	The data contained in the column
lob_information	Contains the LOB information for the column and contains one of the following values: DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1; DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2; DBMS_LCR.INLINE_LOB CONSTANT NUMBER := 3; DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4; DBMS_LCR.LOB_CHUNK CONSTANT NUMBER := 5; DBMS_LCR.LAST_LOB_CHUNK CONSTANT NUMBER := 6;
lob_offset	The LOB offset specified in the number of characters for CLOB columns and the number of bytes for BLOB columns. Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.
lob_operation_size	If lob_information for the LOB is DBMS_LCR.LAST_LOB_CHUNK, then can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB_ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB_TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. If lob_information is not DBMS_LCR.LAST_LOB_CHUNK and for all other operations, is NULL.

109

Rule Types

This chapter describes the types used with rules, rule sets, and evaluation contexts.

This chapter contains the following topic:

- Rule Types

Rule types are used with the following Oracle-supplied PL/SQL packages:

- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_STREAMS_ADM

You can use the DBMS_RULE_ADM package to create and administer rules, rule sets, and evaluation contexts, and you can use the DBMS_RULE package to evaluate rules. When you use Streams, rules determine which changes are captured by a capture process, which events are propagated by a propagation job, and which events are dequeued and applied by an apply process. Also, the DBMS_STREAMS_ADM package creates system-generated rules for use during capture, propagation, and apply.

See Also:

- *Oracle9i Streams*
- Chapter 63, "DBMS_RULE"
- Chapter 64, "DBMS_RULE_ADM"

Rule Types

Table 109–1 DBMS_RULE Types **(Page 1 of 2)**

Data Structure	Description
"RE$ATTRIBUTE_VALUE Type" on page 109-4	Specifies the value of a variable attribute
"RE$ATTRIBUTE_VALUE_LIST Type" on page 109-4	Identifies a list of attribute values used in a rule evaluation context
"RE$COLUMN_VALUE Type" on page 109-5	Specifies the value of a table column
"RE$COLUMN_VALUE_LIST Type" on page 109-5	Identifies a list of column values used in a rule evaluation context
"RE$NAME_ARRAY Type" on page 109-6	Identifies a list of names
"RE$NV_ARRAY Type" on page 109-6	Identifies a list of name-value pairs
"RE$NV_LIST Type" on page 109-6	Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context and the action context for a rule
"RE$NV_NODE Type" on page 109-9	Identifies a name-value pair
"RE$RULE_HIT Type" on page 109-10	Specifies a rule found as a result of evaluation
"RE$RULE_HIT_LIST Type" on page 109-10	Identifies a list of rules found as a result of evaluation
"RE$TABLE_ALIAS Type" on page 109-11	Provides the table corresponding to an alias used in a rule evaluation context
"RE$TABLE_ALIAS_LIST Type" on page 109-11	Identifies a list of table aliases used in a rule evaluation context
"RE$TABLE_VALUE Type" on page 109-12	Specifies the value of a table row using a `ROWID`
"RE$TABLE_VALUE_LIST Type" on page 109-12	Identifies a list of table values used in a rule evaluation context
"RE$VARIABLE_TYPE Type" on page 109-13	Provides the type of a variable used in a rule evaluation context

Table 109–1 DBMS_RULE Types **(Page 2 of 2)**

Data Structure	Description
"RE$VARIABLE_TYPE_LIST Type" on page 109-15	Identifies a list of variables and their types used in a rule evaluation context
"RE$VARIABLE_VALUE Type" on page 109-15	Specifies the value of a variable
"RE$VARIABLE_VALUE_LIST Type" on page 109-15	Identifies a list of variable values used in a rule evaluation context

RE$ATTRIBUTE_VALUE Type

Specifies the value of a variable attribute.

Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE (
   variable_name       VARCHAR2(32),
   attribute_name      VARCHAR2(4000),
   attribute_value     SYS.AnyData);
```

Attributes

Table 109–2 RE$ATTRIBUTE_VALUE Attributes

Attribute	Description
`variable_name`	Specifies the variable used in a rule
`attribute_name`	Specifies the attribute name
`attribute_value`	Specifies the attribute value

RE$ATTRIBUTE_VALUE_LIST Type

Identifies a list of attribute values used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$ATTRIBUTE_VALUE;
```

RE$COLUMN_VALUE Type

Specifies the value of a table column.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE (
   table_alias       VARCHAR2(32),
   column_name       VARCHAR2(4000),
   column_value      SYS.AnyData);
```

Attributes

Table 109–3 RE$COLUMN_VALUE Attributes

Attribute	Description
`table_alias`	Specifies the alias used for the table in a rule
`column_name`	Specifies the column name
`column_value`	Specifies the column value

RE$COLUMN_VALUE_LIST Type

Identifies a list of column values used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE_LIST AS VARRAY(1024) OF SYS.RE$COLUMN_VALUE;
```

RE$NAME_ARRAY Type

Identifies a list of names.

Syntax

```
TYPE SYS.RE$NAME_ARRAY AS VARRAY(1024) OF VARCHAR2(30);
```

RE$NV_ARRAY Type

Identifies a list of name-value pairs.

Syntax

```
TYPE SYS.RE$NV_ARRAY AS VARRAY(1024) OF SYS.RE$NV_NODE;
```

RE$NV_LIST Type

Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context for rule set evaluation and the action context for a rule.

Syntax

```
TYPE SYS.RE$NV_LIST AS OBJECT(
   actx_list    SYS.RE$NV_ARRAY);
```

Attributes

Table 109–4 RE$NV_LIST Attributes

Attribute	Description
actx_list	The list of name-value pairs

RE$NV_LIST Subprograms

This section describes the following member procedures and member functions of the SYS.RE$NV_LIST type:

- ADD_PAIR Member Procedure
- GET_ALL_NAMES Member Function
- GET_VALUE Member Function
- REMOVE_PAIR Member Procedure

ADD_PAIR Member Procedure

Adds a name-value pair to the list of name-value pairs.

Syntax

```
MEMBER PROCEDURE ADD_PAIR(
   name   IN VARCHAR2,
   value  IN SYS.AnyData);
```

Parameters

Table 109–5 ADD_PAIR Procedure Parameters

Parameter	Description
name	The name in the name-value pair being added to the list. If the name already exists in the list, then an error is raised.
value	The value in the name-value pair being added to the list

GET_ALL_NAMES Member Function

Returns a list of all the names in the name-value pair list.

Syntax

```
MEMBER FUNCTION GET_ALL_NAMES RETURN SYS.RE$NAME_ARRAY;
```

GET_VALUE Member Function

Returns the value for the specified name in a name-value pair list.

Syntax

```
MEMBER FUNCTION GET_VALUE(
   name      IN    VARCHAR2)
RETURN SYS.AnyData;
```

Parameters

Table 109–6 GET_VALUE Procedure Parameters

Parameter	Description
name	The name whose value to return

REMOVE_PAIR Member Procedure

Removes the name-value pair with the specified name from the name-value pair list.

Syntax

```
MEMBER PROCEDURE REMOVE_PAIR(
   name     IN    VARCHAR2);
```

Parameters

Table 109–7 REMOVE_PAIR Procedure Parameters

Parameter	Description
name	The name of the pair to remove

RE$NV_NODE Type

Identifies a name-value pair.

Syntax

```
TYPE SYS.RE$NV_NODE (
   nvn_name       VARCHAR2(30),
   nvn_value      SYS.AnyData);
```

Attributes

Table 109–8 RE$NV_NODE Attributes

Attribute	Description
nvn_name	Specifies the name in the name-value pair
nvn_value	Specifies the value in the name-value pair

RE$RULE_HIT Type

Specifies a rule found as a result of an evaluation.

See Also:

- "CREATE_RULE Procedure" on page 64-11
- "ALTER_RULE Procedure" on page 64-5

Syntax

```
TYPE SYS.RE$RULE_HIT (
   rule_name              VARCHAR2(61),
   rule_action_context    RE$NV_LIST);
```

Attributes

Table 109–9 RE$RULE_HIT Attributes

Attribute	Description
`rule_name`	The rule name in the form *`schema_name.rule_name`*. For example, a rule named `employee_rule` in the `hr` schema is returned in the form `hr.employee_rule`.
`rule_action_context`	The rule action context as specified in the `CREATE_RULE` or `ALTER_RULE` procedure of the `DBMS_RULE_ADM` package

RE$RULE_HIT_LIST Type

Identifies a list of rules found as a result of an evaluation.

Syntax

```
TYPE SYS.RE$RULE_HIT_LIST AS VARRAY(1024) OF SYS.RE$RULE_HIT;
```

RE$TABLE_ALIAS Type

Provides the table corresponding to an alias used in a rule evaluation context. A specified table name must satisfy the schema object naming rules.

> **See Also:** *Oracle9i SQL Reference* for information about schema object naming rules

Syntax

```
TYPE SYS.RE$TABLE_ALIAS IS OBJECT(
   table_alias   VARCHAR2(32),
   table_name    VARCHAR2(194));
```

Attributes

Table 109–10 RE$TABLE_ALIAS Attributes

Attribute	Description
`table_alias`	The alias used for the table in a rule
`table_name`	The table name referred to by the alias. A synonym can be specified. The table name is resolved in the evaluation context schema. The format is the following: *schema_name.table_name* For example, if the *schema_name* is `hr` and the *table_name* is `employees`, then enter the following: `hr.employees`

RE$TABLE_ALIAS_LIST Type

Identifies a list of table aliases used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$TABLE_ALIAS_LIST AS VARRAY(1024) OF SYS.RE$TABLE_ALIAS;
```

RE$TABLE_VALUE Type

Specifies the value of a table row using a ROWID.

Syntax

```
TYPE SYS.RE$TABLE_VALUE(
   table_alias       VARCHAR2(32),
   table_rowid       VARCHAR2(18));
```

Attributes

Table 109–11 RE$TABLE_VALUE Attributes

Attribute	Description
table_alias	Specifies the alias used for the table in a rule
table_rowid	Specifies the rowid for the table row

RE$TABLE_VALUE_LIST Type

Identifies a list of table values used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$TABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$TABLE_VALUE;
```

RE$VARIABLE_TYPE Type

Provides the type of a variable used in a rule evaluation context. A specified variable name must satisfy the schema object naming rules.

> **See Also:** *Oracle9i SQL Reference* for information about schema object naming rules

Syntax

```
TYPE SYS.RE$VARIABLE_TYPE (
   variable_name             VARCHAR2(32),
   variable_type             VARCHAR2(4000),
   variable_value_function   VARCHAR2(228),
   variable_method_function  VARCHAR2(228));
```

Attributes

Table 109–12 RE$VARIABLE_TYPE Attributes

Attribute	Description
`variable_name`	The variable name used in a rule
`variable_type`	The type that is resolved in the evaluation context schema. Any valid Oracle built-in datatype, user-defined type, or Oracle-supplied type can be specified. See the *Oracle9i SQL Reference* for more information about these types.
`variable_value_function`	A value function that can be specified for implicit variables. A synonym can be specified. The function name is resolved in the evaluation context schema. See the "Usage Notes" for more information.
`variable_method_function`	Specifies a value function, which can return the result of a method invocation. Specifying such a function can speed up evaluation, if there are many simple rules that invoke the method on the variable. The function can be a synonym or a remote function. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context. See the "Usage Notes" for more information.

Usage Notes

The functions for both the for the `variable_value_function` parameter and `variable_method_function` parameter have the following format:

schema_name.package_name.function_name@dblink

For example, if the *schema_name* is `hr`, the *package_name* is `var_pac`, the *function_name* is `func_value`, and the *dblink* is `dbs1.net`, then enter the following:

```
hr.var_pac.func_value@dbs1.net
```

The following sections describe the signature of the functions.

Signature for variable_value_function

The function must have the following signature:

```
FUNCTION variable_value_func(
  evaluation_context_schema  IN VARCHAR2,
  evaluation_context_name    IN VARCHAR2,
  variable_name              IN VARCHAR2,
  event_context              IN SYS.RE$NV_LIST )
RETURN SYS.RE$VARIABLE_VALUE;
```

Signature for variable_method_function

This function must have the following signature:

```
FUNCTION variable_method_function(
  evaluation_context_schema  IN VARCHAR2,
  evaluation_context_name    IN VARCHAR2,
  variable_value             IN SYS.RE$VARIABLE_VALUE,
  method_name                IN VARCHAR2,
  event_context              IN SYS.RE$NV_LIST)
RETURN SYS.RE$ATTRIBUTE_VALUE;
```

RE$VARIABLE_TYPE_LIST Type

Identifies a list of variables and their types used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$VARIABLE_TYPE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_TYPE;
```

RE$VARIABLE_VALUE Type

Specifies the value of a variable.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE (
   variable_name       VARCHAR2(32),
   variable_data       SYS.AnyData);
```

Attributes

Table 109–13 RE$VARIABLE_VALUE Attributes

Attribute	Description
`variable_name`	Specifies the variable name used in a rule
`variable_data`	Specifies the data for the variable value

RE$VARIABLE_VALUE_LIST Type

Identifies a list of variable values used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_VALUE;
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W