Oracle9i

Supplied PL/SQL Packages and Types Reference

Release 1 (9.0.1)

July 2001
Part No. A89852-02
### Contents

**Send Us Your Comments** .................................................................................................................. xvii

**Preface**.................................................................................................................................................. xix
- Audience ................................................................................................................................................ xx
- Related Documentation .......................................................................................................................... xx
- Conventions .......................................................................................................................................... xxi
- Documentation Accessibility ................................................................................................................ xxiii

**What’s New in Supplied PL/SQL Packages and Types?**................................................................. xxv
- Oracle9i Release 1 (9.0.1) New Features in Supplied PL/SQL Packages and Types .............. xxvi
- Oracle8i Release 2 (8.1.6) New Features in Supplied PL/SQL Packages ......................... xxvii
- Oracle8i Release 1 (8.1.5) New Features in Supplied PL/SQL Packages ........................ xxvii

1 **Introduction**
- Package Overview ............................................................................................................................ 1-2
- Abbreviations for Datetime and Interval Datatypes ................................................................. 1-7
- Summary of Oracle Supplied PL/SQL Packages ........................................................................ 1-7
- Summary of Subprograms in Supplemental Packages ............................................................. 1-14

2 **DBMS_ALERT**
- Security, Constants, and Errors for DBMS_ALERT ................................................................. 2-2
- Using Alerts ....................................................................................................................................... 2-3
- Summary of DBMS_ALERT Subprograms ................................................................................. 2-4
10 DBMS_DEFER
Summary of DBMS_DEFER Subprograms ................................................................. 10-2

11 DBMS_DEFER_QUERY
Summary of DBMS_DEFER_QUERY Subprograms ................................................... 11-2

12 DBMS_DEFER_SYS
Summary of DBMS_DEFER_SYS Subprograms ....................................................... 12-2

13 DBMS_DESCRIBE
Security, Types, and Errors for DBMS_DESCRIBE ............................................. 13-2
Summary of DBMS_DESCRIBE Subprograms ......................................................... 13-2

14 DBMS_DISTRIBUTED_TRUST_ADMIN
Requirements ........................................................................................................... 14-2
Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms ......................... 14-2

15 DBMS_FGA
Summary of DBMS_FGA Subprogram .................................................................... 15-2

16 DBMS_FLASHBACK
DBMS_FLASHBACK Error Messages ........................................................................ 16-3
DBMS_FLASHBACK Example .................................................................................. 16-3
Summary of DBMS_FLASHBACK Subprograms ...................................................... 16-6

17 DBMS_HS_PASSTHROUGH
Security .................................................................................................................... 17-2
Summary of DBMS_HS_PASSTHROUGH Subprograms .......................................... 17-2

18 DBMS_IOT
Summary of DBMS_IOT Subprograms .................................................................... 18-2
19 DBMS_JOB
Requirements.................................................................................................................. 19-2
Using the DBMS_JOB Package with Oracle Real Application Clusters ...................... 19-2
Summary of DBMS_JOB Subprograms ........................................................................ 19-3

20 DBMS_LDAP
Exception Summary ........................................................................................................ 20-2
Summary of Data Types ................................................................................................. 20-3
Summary of DBMS_LDAP Subprograms ...................................................................... 20-4

21 DBMS_LIBCACHE
Requirements ................................................................................................................. 21-2
Summary of DBMS_LIBCACHE Subprograms ............................................................... 21-2

22 DBMS_LOB
LOB Locators for DBMS_LOB ....................................................................................... 22-2
Datatypes, Constants, and Exceptions for DBMS_LOB ................................................ 22-2
Security for DBMS_LOB ................................................................................................. 22-4
Rules and Limitations for DBMS_LOB ........................................................................ 22-5
Temporary LOBs ........................................................................................................... 22-9
Summary of DBMS_LOB Subprograms ...................................................................... 22-13

23 DBMS_LOCK
Requirements, Security, and Constants for DBMS_LOCK ........................................... 23-2
Summary of DBMS_LOCK Subprograms ....................................................................... 23-3

24 DBMS_LOGMNR
DBMS_LOGMNR Constants ......................................................................................... 24-2
Extracting Data Values From Redo Log Files ............................................................ 24-3
Example of Using DBMS_LOGMNR .............................................................................. 24-3
Summary of DBMS_LOGMNR Subprograms ............................................................... 24-4
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Package Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>DBMS_LOGMNR_CDC_PUBLISH</td>
<td>Publishing Change Data</td>
<td>25-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_LOGMNR_CDC_PUBLISH Subprograms</td>
<td>25-2</td>
</tr>
<tr>
<td>26</td>
<td>DBMS_LOGMNR_CDC_SUBSCRIBE</td>
<td>Subscribing to Change Data</td>
<td>26-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_LOGMNR_CDC_SUBSCRIBE Subprograms</td>
<td>26-2</td>
</tr>
<tr>
<td>27</td>
<td>DBMS_LOGMNR_D</td>
<td>Extracting a Dictionary to the Redo Log Files</td>
<td>27-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extracting a Dictionary to a Flat File</td>
<td>27-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Examples of Using DBMS_LOGMNR_D.BUILD</td>
<td>27-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_LOGMNR_D Subprograms</td>
<td>27-3</td>
</tr>
<tr>
<td>28</td>
<td>DBMS_METADATA</td>
<td>Summary of DBMS_METADATA Subprograms</td>
<td>28-2</td>
</tr>
<tr>
<td>29</td>
<td>DBMS_MVIEW</td>
<td>Summary of DBMS_MVIEW Subprograms</td>
<td>29-2</td>
</tr>
<tr>
<td>30</td>
<td>DBMS_OBFUSCATION_TOOLKIT</td>
<td>Overview of Key Management</td>
<td>30-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_OBFUSCATION Subprograms</td>
<td>30-5</td>
</tr>
<tr>
<td>31</td>
<td>DBMS_ODCI</td>
<td>Summary of DBMS_ODCI Subprograms</td>
<td>31-2</td>
</tr>
<tr>
<td>32</td>
<td>DBMS_OFFLINE_OG</td>
<td>Summary of DBMS_OFFLINE_OG Subprograms</td>
<td>32-2</td>
</tr>
<tr>
<td>33</td>
<td>DBMS_OFFLINE_SNAPSHOT</td>
<td>Summary of DBMS_OFFLINE_SNAPSHOT Subprograms</td>
<td>33-2</td>
</tr>
<tr>
<td>Chapter</td>
<td>Module</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>34</td>
<td>DBMS_OLEAP</td>
<td>Requirements</td>
<td>34-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error Messages</td>
<td>34-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_OLEAP Subprograms</td>
<td>34-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS_OLEAP Interface Views</td>
<td>34-21</td>
</tr>
<tr>
<td>35</td>
<td>DBMS_ORACLE_TRACE_AGENT</td>
<td>Security</td>
<td>35-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_ORACLE_TRACE_AGENT Subprograms</td>
<td>35-2</td>
</tr>
<tr>
<td>36</td>
<td>DBMS_ORACLE_TRACE_USER</td>
<td>Summary of DBMS_ORACLE_TRACE_USER Subprograms</td>
<td>36-2</td>
</tr>
<tr>
<td>37</td>
<td>DBMS_OUTLN</td>
<td>Requirements and Security for DBMS_OUTLN</td>
<td>37-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_OUTLN Subprograms</td>
<td>37-2</td>
</tr>
<tr>
<td>38</td>
<td>DBMS_OUTLN_EDIT</td>
<td>Summary of DBMS_OUTLN_EDIT Subprograms</td>
<td>38-2</td>
</tr>
<tr>
<td>39</td>
<td>DBMS_OUTPUT</td>
<td>Security, Errors, and Types for DBMS_OUTPUT</td>
<td>39-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Using DBMS_OUTPUT</td>
<td>39-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_OUTPUT Subprograms</td>
<td>39-3</td>
</tr>
<tr>
<td>40</td>
<td>DBMS_PCLXUTIL</td>
<td>Using DBMS_PCLXUTIL</td>
<td>40-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limitations</td>
<td>40-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of DBMS_PCLXUTIL Subprograms</td>
<td>40-3</td>
</tr>
<tr>
<td>41</td>
<td>DBMS_PIPE</td>
<td>Public Pipes, Private Pipes, and Pipe Uses</td>
<td>41-2</td>
</tr>
<tr>
<td>Module ID</td>
<td>Module Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>DBMS_REPCAT_ADMIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>DBMS_REPCAT_INSTANTIATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>DBMS_REPCAT_RGT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>DBMS_REPUTIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>DBMS_RESOURCE_MANAGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>DBMS_RESOURCE_MANAGER_PRIVS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>DBMS_RESUMABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>DBMS_RLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>DBMS_ROWID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DBMS_REPCAT_ADMIN**
Summary of DBMS_REPCAT_ADMIN Subprograms ......................................................... 49-2

**DBMS_REPCAT_INSTANTIATE**
Summary of DBMS_REPCAT_INSTANTIATE Subprograms .................................................. 50-2

**DBMS_REPCAT_RGT**
Summary of DBMS_REPCAT_RGT Subprograms ............................................................... 51-2

**DBMS_REPUTIL**
Summary of DBMS_REPUTIL Subprograms ........................................................................ 52-2

**DBMS_RESOURCE_MANAGER**
Requirements .................................................................................................................. 53-2
Summary of DBMS_RESOURCE_MANAGER Subprograms .................................................... 53-2

**DBMS_RESOURCE_MANAGER_PRIVS**
Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms ........................................... 54-2

**DBMS_RESUMABLE**
Summary of DBMS_RESUMABLE Subprograms ................................................................ 55-2

**DBMS_RLS**
Dynamic Predicates ........................................................................................................ 56-2
Security .......................................................................................................................... 56-3
Usage Notes ..................................................................................................................... 56-3
Summary of DBMS_RLS Subprograms .............................................................................. 56-3

**DBMS_ROWID**
Usage Notes ..................................................................................................................... 57-2
Requirements ................................................................................................................... 57-3
ROWID Types ................................................................................................................... 57-3
DBMS_STATS
Using DBMS_STATS ................................................................. 63-2
Setting or Getting Statistics ................................................... 63-3
Gathering Optimizer Statistics ................................................. 63-4
Transferring Statistics ............................................................ 63-4
Summary of DBMS_STATS Subprograms ................................. 63-5

DBMS_TRACE
Requirements, Restrictions, and Constants for DBMS_TRACE ..................................................... 64-2
Using DBMS_TRACE ................................................................. 64-2
Summary of DBMS_TRACE Subprograms .................................. 64-5

DBMS_TRANSACTION
Requirements ........................................................................... 65-2
Summary of DBMS_TRANSACTION Subprograms ............................. 65-2

DBMS_TRANSFORM
Summary of DBMS_TRANSFORM Subprograms ............................... 66-2

DBMS_TTS
Exceptions ................................................................................ 67-2
Summary of DBMS_TTS Subprograms .......................................... 67-2

DBMS_TYPES
Constants for DBMS_TYPES ...................................................... 68-2

DBMS.Utility
Requirements and Types for DBMS.Utility ................................ 69-2
Summary of DBMS_Utility Subprograms ...................................... 69-2

DBMS_WM
Summary of DBMS_WM Subprograms .......................................... 70-2
71 DBMS_XMLGEN
Summary of DBMS_XMLGEN Subprograms ................................................................. 71-2

72 DBMS_XMLQUERY
Summary of DBMS_XMLQUERY Subprograms ......................................................... 72-2

73 DBMS_XMLSAVE
Summary of DBMS_XMLSAVE Subprograms .......................................................... 73-2

74 DEBUG_EXTPROC
Requirements and Installation Notes for DEBUG_EXTPROC .......................... 74-2
Using DEBUG_EXTPROC .................................................................................... 74-2
Summary of DBMS_EXTPROC Subprograms ....................................................... 74-3

75 UTL_COLL
Summary of UTL_COLL Subprograms ................................................................. 75-2

76 UTL_ENCODE
Summary of UTL_ENCODE Subprograms .............................................................. 76-2

77 UTL_FILE
Security .................................................................................................................. 77-2
File Ownership and Protections ............................................................................ 77-2
Types ....................................................................................................................... 77-3
Exceptions .............................................................................................................. 77-4
Summary of UTL_FILE Subprograms ................................................................. 77-4

78 UTL_HTTP
UTL_HTTP Constants, Types and Flow ................................................................. 78-2
UTL_HTTP Exceptions .......................................................................................... 78-10
UTL_HTTP Examples ............................................................................................ 78-11
Summary of UTL_HTTP Subprograms ................................................................. 78-15
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Package</th>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>UTL_INADDR</td>
<td>Exceptions</td>
<td>79-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_INADDR Subprograms</td>
<td>79-2</td>
</tr>
<tr>
<td>80</td>
<td>UTL_RAW</td>
<td>Usage Notes</td>
<td>80-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_RAW Subprograms</td>
<td>80-2</td>
</tr>
<tr>
<td>81</td>
<td>UTL_REF</td>
<td>Requirements</td>
<td>81-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Datatypes, Exceptions, and Security for UTL_REF</td>
<td>81-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_REF Subprograms</td>
<td>81-4</td>
</tr>
<tr>
<td>82</td>
<td>UTL_SMTP</td>
<td>Example</td>
<td>82-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exceptions, Limitations, and Reply Codes</td>
<td>82-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_SMTP Subprograms</td>
<td>82-6</td>
</tr>
<tr>
<td>83</td>
<td>UTL_TCP</td>
<td>Exceptions</td>
<td>83-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example</td>
<td>83-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_TCP Subprograms</td>
<td>83-4</td>
</tr>
<tr>
<td>84</td>
<td>UTL_URL</td>
<td>Introduction to the UTL_URL Package</td>
<td>84-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTL_URL Exceptions</td>
<td>84-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of UTL_URL Subprograms</td>
<td>84-3</td>
</tr>
<tr>
<td>85</td>
<td>ANYDATA TYPE</td>
<td>Construction</td>
<td>85-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summary of ANYDATA Subprograms</td>
<td>85-2</td>
</tr>
</tbody>
</table>
86 ANYDATASET TYPE

Construction .................................................................................................................. 86-2
Summary of ANYDATASET Subprograms .................................................................... 86-2

87 ANYTYPE TYPE

Summary of ANYTYPE Subprograms .......................................................................... 87-2

Index
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: Information Development Department Manager
- Postal service:
  Oracle Corporation
  Information Development Department
  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.
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 discusses the following:

- **Audience**
- **Related Documentation**
- **Conventions**
- **Documentation Accessibility**

**See Also:** For information about Java packages, please refer to Oracle9i Supplied Java Packages Reference.
**Audience**

This manual is intended for programmers, systems analysts, project managers, and others interested in the development and tuning of database applications.

This manual assumes you have a working knowledge of application programming and that you are familiar with the use of structured query language (SQL) to access information in relational database systems.

Certain sections of this manual also assume a knowledge of the basic concepts of object-oriented programming.

**Related Documentation**

For more information, see the following manuals in the Oracle9i documentation set:

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

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://technet.oracle.com/membership/index.htm

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://technet.oracle.com/docs/index.htm
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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.</td>
<td>When you specify this clause, you create an index-organized table.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders.</td>
<td>Oracle9i Database Concepts</td>
</tr>
<tr>
<td><strong>UPPERCASE monospace</strong></td>
<td>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.</td>
<td>Ensure that the recovery catalog and target database do not reside on the same disk.</td>
</tr>
<tr>
<td><strong>lowercase monospace</strong></td>
<td>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.</td>
<td>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.</td>
</tr>
</tbody>
</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:

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

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

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Brackets enclose one or more optional items. Do not enter the brackets.</td>
<td><code>DECIMAL (digits [, precision ])</code></td>
</tr>
<tr>
<td>{ }</td>
<td>Braces enclose two or more items, one of which is required. Do not enter the braces.</td>
<td>`{ENABLE</td>
</tr>
</tbody>
</table>
| |           | 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:                              | `CREATE TABLE ... AS subquery;`              |
|            | ■ That we have omitted parts of the code that are not directly related to the example | `SELECT col1, col2, ..., coln FROM employees;` |
|            | ■ That you can repeat a portion of the code                              |                                             |
| .          | Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example. | `acctbal NUMBER(11,2);`                     |
|            | Other notation You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown. | `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`                   |
Documentation Accessibility

Oracle’s goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. 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 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/

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 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**
Oracle9i 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:
Oracle8i 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
- DBMS_PROFILER
- DBMS_REPAIR
- DBMS_RESOURCE_MANAGER
- DBMS_ROWID
- DBMS_SQL
- DBMSUTILITY
- UTL_HTTP

Oracle8i Release 1 (8.1.5) New Features in Supplied PL/SQL Packages

This book was new for release 8.1.5.
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:

- Package Overview
- Summary of Oracle Supplied PL/SQL Packages
- Summary of Subprograms in Supplemental Packages

**See Also:** For information on how to create your own packages, see the Oracle9i Application Developer’s Guide - Fundamentals.
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];
```

1-2  Supplied PL/SQL Packages and Types Reference
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: For more information on creating new packages, see PL/SQL User’s Guide and Reference and Oracle9i Application Developer’s Guide - Fundamentals. For more information on storing and executing packages, see Oracle9i Database Concepts.

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. This distinction allows you to 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.

```sql
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:

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

---

**Referencing Package Contents**

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

```sql
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.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>TSTZ</td>
</tr>
<tr>
<td>TIMESTAMP LOCAL TIME ZONE</td>
<td>TSLTZ</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>IYM</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>IDS</td>
</tr>
</tbody>
</table>

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.
Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_ALERT</td>
<td>Provides support for the asynchronous notification of database events.</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO</td>
<td>Lets you register an application name with the database for auditing or performance tracking purposes.</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>DBMS_AQ</td>
<td>Lets you add a message (of a predefined object type) onto a queue or to dequeue a message.</td>
<td>Chapter 4</td>
</tr>
<tr>
<td>DBMS_AQADM</td>
<td>Lets you perform administrative functions on a queue or queue table for messages of a predefined object type.</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>DBMS_AQEIM</td>
<td>Provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP.</td>
<td>Chapter 6</td>
</tr>
<tr>
<td>DBMS_BACKUP_RESTORE</td>
<td>Normalizes filenames on Windows NT platforms.</td>
<td>Chapter 7</td>
</tr>
<tr>
<td>DBMS_DDL</td>
<td>Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs.</td>
<td>Chapter 8</td>
</tr>
<tr>
<td>DBMS_DEBUG</td>
<td>Implements server-side debuggers and provides a way to debug server-side PL/SQL program units.</td>
<td>Chapter 9</td>
</tr>
<tr>
<td>DBMS_DEFER</td>
<td>Provides the user interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.</td>
<td>Chapter 10</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY</td>
<td>Permits querying the deferred remote procedure calls (RPC) queue data that is not exposed through views. Requires the Distributed Option.</td>
<td>Chapter 11</td>
</tr>
<tr>
<td>DBMS_DEFER_SYS</td>
<td>Provides the system administrator interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.</td>
<td>Chapter 12</td>
</tr>
<tr>
<td>DBMS_DESCRIBE</td>
<td>Describes the arguments of a stored procedure with full name translation and security checking.</td>
<td>Chapter 13</td>
</tr>
<tr>
<td>DBMS_DISTRIBUTED_TRUST_ADMIN</td>
<td>Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted.</td>
<td>Chapter 14</td>
</tr>
<tr>
<td>DBMS_FGA</td>
<td>Provides fine-grained security functions.</td>
<td>Chapter 15</td>
</tr>
</tbody>
</table>
### Table 1–1  Summary of Oracle Supplied PL/SQL Packages (Cont.)

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DMBS_FLASHBACK</strong></td>
<td>Lets you flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN).</td>
<td>Chapter 16</td>
</tr>
<tr>
<td><strong>DBMS_HS_PASSTHROUGH</strong></td>
<td>Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems.</td>
<td>Chapter 17</td>
</tr>
<tr>
<td><strong>DBMS_IOT</strong></td>
<td>Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command.</td>
<td>Chapter 18</td>
</tr>
<tr>
<td><strong>DBMS_JOB</strong></td>
<td>Lets you schedule administrative procedures that you want performed at periodic intervals; it is also the interface for the job queue.</td>
<td>Chapter 19</td>
</tr>
<tr>
<td><strong>DBMS_LDAP</strong></td>
<td>Provides functions and procedures to access data from LDAP servers.</td>
<td>Chapter 20</td>
</tr>
<tr>
<td><strong>DBMS_LIBCACHE</strong></td>
<td>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.</td>
<td>Chapter 21</td>
</tr>
<tr>
<td><strong>DBMS_LOB</strong></td>
<td>Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read-write), and BFILES (read-only).</td>
<td>Chapter 22</td>
</tr>
<tr>
<td><strong>DBMS_LOCK</strong></td>
<td>Lets you request, convert and release locks through Oracle Lock Management services.</td>
<td>Chapter 23</td>
</tr>
<tr>
<td><strong>DBMS_LOGMNR</strong></td>
<td>Provides functions to initialize and run the log reader.</td>
<td>Chapter 24</td>
</tr>
<tr>
<td><strong>DBMS_LOGMNR_CDC_PUBLISH</strong></td>
<td>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.</td>
<td>Chapter 25</td>
</tr>
<tr>
<td><strong>DBMS_LOGMNR_CDC_SUBSCRIBE</strong></td>
<td>Lets you view and query the change data that was captured and published with the DBMS_LOGMNR_CDC_PUBLISH package.</td>
<td>Chapter 26</td>
</tr>
<tr>
<td><strong>DBMS_LOGMNR_D</strong></td>
<td>Queries the dictionary tables of the current database, and creates a text based file containing their contents.</td>
<td>Chapter 27</td>
</tr>
<tr>
<td><strong>DBMS_METADATA</strong></td>
<td>Lets callers easily retrieve complete database object definitions (metadata) from the dictionary.</td>
<td>Chapter 28</td>
</tr>
</tbody>
</table>
## Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_MVIEW</td>
<td>Lets you refresh snapshots that are not part of the same refresh group and purge logs. DBMS_MVIEW_SNAPSHOT is a synonym.</td>
<td>Chapter 29</td>
</tr>
<tr>
<td>DBMS_GBPUSCATION_TOOLKIT</td>
<td>Provides procedures for Data Encryption Standards.</td>
<td>Chapter 30</td>
</tr>
<tr>
<td>DBMS_GDCI</td>
<td>Returns the CPU cost of a user function based on the elapsed time of the function.</td>
<td>Chapter 31</td>
</tr>
<tr>
<td>DBMS_OFFLINE_OG</td>
<td>Provides public APIs for offline instantiation of master groups.</td>
<td>Chapter 32</td>
</tr>
<tr>
<td>DBMS_OFFLINE_SNAPSHOT</td>
<td>Provides public APIs for offline instantiation of snapshots.</td>
<td>Chapter 33</td>
</tr>
<tr>
<td>DBMS_DLMAP</td>
<td>Provides procedures for summaries, dimensions, and query rewrites.</td>
<td>Chapter 34</td>
</tr>
<tr>
<td>DBMS_ORACLE_TRACE_AGENT</td>
<td>Provides client callable interfaces to the Oracle TRACE instrumentation within the Oracle Server.</td>
<td>Chapter 35</td>
</tr>
<tr>
<td>DBMS_ORACLE_TRACE_USER</td>
<td>Provides public access to the Oracle release 7 Server Oracle TRACE instrumentation for the calling user.</td>
<td>Chapter 36</td>
</tr>
<tr>
<td>DBMS.Override.IN</td>
<td>Provides the interface for procedures and functions associated with management of stored outlines. Synonymous with OUTLN_PKG</td>
<td>Chapter 37</td>
</tr>
<tr>
<td>DBMS.Override.IN_EDIT</td>
<td>Lets you edit an invoker’s rights package.</td>
<td>Chapter 38</td>
</tr>
<tr>
<td>DBMS_OUTPUT</td>
<td>Accumulates information in a buffer so that it can be retrieved out later.</td>
<td>Chapter 39</td>
</tr>
<tr>
<td>DBMS_PCLXUTIL</td>
<td>Provides intra-partition parallelism for creating partition-wise local indexes.</td>
<td>Chapter 40</td>
</tr>
<tr>
<td>DBMS_PIPE</td>
<td>Provides a DBMS pipe service which enables messages to be sent between sessions.</td>
<td>Chapter 41</td>
</tr>
<tr>
<td>DBMS_PROFILER</td>
<td>Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks.</td>
<td>Chapter 42</td>
</tr>
<tr>
<td>DBMS_RANDOM</td>
<td>Provides a built-in random number generator.</td>
<td>Chapter 43</td>
</tr>
<tr>
<td>DBMS_RECTIFIER_DIFF</td>
<td>Provides APIs used to detect and resolve data inconsistencies between two replicated sites.</td>
<td>Chapter 44</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
<td>Documentation</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DBMS_REDEFINITION</td>
<td>Lets you perform an online reorganization of tables.</td>
<td>Chapter 45</td>
</tr>
<tr>
<td>DBMS_REFRESH</td>
<td>Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time. Requires the Distributed Option.</td>
<td>Chapter 46</td>
</tr>
<tr>
<td>DBMS_REPAIR</td>
<td>Provides data corruption repair procedures.</td>
<td>Chapter 47</td>
</tr>
<tr>
<td>DBMS_REPCAT</td>
<td>Provides routines to administer and update the replication catalog and environment. Requires the Replication Option.</td>
<td>Chapter 48</td>
</tr>
<tr>
<td>DBMS_REPCAT_ADMIN</td>
<td>Lets you create users with the privileges needed by the symmetric replication facility. Requires the Replication Option.</td>
<td>Chapter 49</td>
</tr>
<tr>
<td>DBMS_REPCAT_INSTANTIATE</td>
<td>Instantiates deployment templates. Requires the Replication Option.</td>
<td>Chapter 50</td>
</tr>
<tr>
<td>DBMS_REPCAT_RGT</td>
<td>Controls the maintenance and definition of refresh group templates. Requires the Replication Option.</td>
<td>Chapter 51</td>
</tr>
<tr>
<td>DBMS_REPUTIL</td>
<td>Provides routines to generate shadow tables, triggers, and packages for table replication.</td>
<td>Chapter 52</td>
</tr>
<tr>
<td>DBMS_RESOURCE_MANAGER</td>
<td>Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema.</td>
<td>Chapter 53</td>
</tr>
<tr>
<td>DBMS_RESOURCE_MANAGER_PRIVS</td>
<td>Maintains privileges associated with resource consumer groups.</td>
<td>Chapter 54</td>
</tr>
<tr>
<td>DBMS_RESUMABLE</td>
<td>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.</td>
<td>Chapter 55</td>
</tr>
<tr>
<td>DBMS_RLS</td>
<td>Provides row level security administrative interface.</td>
<td>Chapter 56</td>
</tr>
<tr>
<td>DBMS_ROWID</td>
<td>Provides procedures to create rowids and to interpret their contents.</td>
<td>Chapter 57</td>
</tr>
<tr>
<td>DBMS_SESSION</td>
<td>Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures.</td>
<td>Chapter 58</td>
</tr>
</tbody>
</table>
### Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_SHARED_POOL</td>
<td>Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism.</td>
<td>Chapter 59</td>
</tr>
<tr>
<td>DBMS_SNAPSHOT</td>
<td>Synonym for DBMS_MVIEW</td>
<td>Chapter 29</td>
</tr>
<tr>
<td>DBMS_SPACE</td>
<td>Provides segment space information not available through standard SQL.</td>
<td>Chapter 60</td>
</tr>
<tr>
<td>DBMS_SPACE_ADMIN</td>
<td>Provides tablespace and segment space administration not available through the standard SQL.</td>
<td>Chapter 61</td>
</tr>
<tr>
<td>DBMS_SQL</td>
<td>Lets you use dynamic SQL to access the database.</td>
<td>Chapter 62</td>
</tr>
<tr>
<td>DBMS_STANDARD</td>
<td>Provides language facilities that help your application interact with Oracle.</td>
<td>Refer to Note #1</td>
</tr>
<tr>
<td>DBMS_STATS</td>
<td>Provides a mechanism for users to view and modify optimizer statistics gathered for database objects.</td>
<td>Chapter 63</td>
</tr>
<tr>
<td>DBMS_TRACE</td>
<td>Provides routines to start and stop PL/SQL tracing.</td>
<td>Chapter 64</td>
</tr>
<tr>
<td>DBMS_TRANSACTION</td>
<td>Provides access to SQL transaction statements from stored procedures and monitors transaction activities.</td>
<td>Chapter 65</td>
</tr>
<tr>
<td>DBMS_TRANSFORM</td>
<td>Provides an interface to the message format transformation features of Oracle Advanced Queuing.</td>
<td>Chapter 66</td>
</tr>
<tr>
<td>DBMS_TTS</td>
<td>Checks if the transportable set is self-contained.</td>
<td>Chapter 67</td>
</tr>
<tr>
<td>DBMS_TYPES</td>
<td>Consists of constants, which represent the built-in and user-defined types.</td>
<td>Chapter 68</td>
</tr>
<tr>
<td>DBMS_UTILITY</td>
<td>Provides various utility routines.</td>
<td>Chapter 69</td>
</tr>
<tr>
<td>DBMS_WM</td>
<td>Describes how to use the the programming interface to Oracle Database Workspace Manager to work with long transactions.</td>
<td>Chapter 70</td>
</tr>
<tr>
<td>DBMS_XMLGEN</td>
<td>Converts the results of a SQL query to a canonical XML format.</td>
<td>Chapter 71</td>
</tr>
<tr>
<td>DBMS_XMLQUERY</td>
<td>Provides database-to-XMLType functionality.</td>
<td>Chapter 72</td>
</tr>
<tr>
<td>DBMS_XMLSAVE</td>
<td>Provides XML-to-database-type functionality.</td>
<td>Chapter 73</td>
</tr>
</tbody>
</table>
### Table 1–1  Summary of Oracle Supplied PL/SQL Packages (Cont.)

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG_EXTPROC</td>
<td>Lets you debug external procedures on platforms with debuggers that can attach to a running process.</td>
<td>Chapter 74</td>
</tr>
<tr>
<td>OUTLN_PKG</td>
<td>Synonym of DBMS_OUTLN.</td>
<td>Chapter 37</td>
</tr>
<tr>
<td>PLITBLIM</td>
<td>Handles index-table operations.</td>
<td>Refer to Note #1</td>
</tr>
<tr>
<td>SDO_CS (refer to Note #2)</td>
<td>Provides functions for coordinate system transformation.</td>
<td>Oracle Spatial User’s Guide and Reference</td>
</tr>
<tr>
<td>SDO_GEOB (refer to Note #2)</td>
<td>Provides functions implementing geometric operations on spatial objects.</td>
<td>Oracle Spatial User’s Guide and Reference</td>
</tr>
<tr>
<td>SDO_LRS (refer to Note #2)</td>
<td>Provides functions for linear referencing system support.</td>
<td>Oracle Spatial User’s Guide and Reference</td>
</tr>
<tr>
<td>SDO_MIGRATE (refer to Note #2)</td>
<td>Provides functions for migrating spatial data from previous releases.</td>
<td>Oracle Spatial User’s Guide and Reference</td>
</tr>
<tr>
<td>SDO_TUNE (refer to Note #2)</td>
<td>Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in Oracle Spatial.</td>
<td>Oracle Spatial User’s Guide and Reference</td>
</tr>
<tr>
<td>STANDARD</td>
<td>Declares types, exceptions, and subprograms which are available automatically to every PL/SQL program.</td>
<td>Refer to Note #1</td>
</tr>
<tr>
<td>UTL_COLL</td>
<td>Enables PL/SQL programs to use collection locators to query and update.</td>
<td>Chapter 75</td>
</tr>
<tr>
<td>UTL_ENCODE</td>
<td>Provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts.</td>
<td>Chapter 76</td>
</tr>
<tr>
<td>UTL_FILE</td>
<td>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.</td>
<td>Chapter 77</td>
</tr>
<tr>
<td>UTL_HTTP</td>
<td>Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges.</td>
<td>Chapter 78</td>
</tr>
<tr>
<td>UTL_INADDR</td>
<td>Provides a procedure to support internet addressing.</td>
<td>Chapter 79</td>
</tr>
</tbody>
</table>
Table 1–1 Summary of Oracle Supplied PL/SQL Packages (Cont.)

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTL_PG</td>
<td>Provides functions for converting COBOL numeric data into Oracle numbers and Oracle numbers into COBOL numeric data.</td>
<td>Oracle Procedural Gateway for APPC User's Guide</td>
</tr>
<tr>
<td>UTL_RAW</td>
<td>Provides SQL functions for RAW datatypes that concat, substr, etc. to and from RAWS.</td>
<td>Chapter 80</td>
</tr>
<tr>
<td>UTL_REF</td>
<td>Enables a PL/SQL program to access an object by providing a reference to the object.</td>
<td>Chapter 81</td>
</tr>
<tr>
<td>UTL_SMTP</td>
<td>Provides PL/SQL functionality to send emails.</td>
<td>Chapter 82</td>
</tr>
<tr>
<td>UTL_TCP</td>
<td>Provides PL/SQL functionality to support simple TCP/IP-based communications between servers and the outside world.</td>
<td>Chapter 83</td>
</tr>
<tr>
<td>UTL_URL</td>
<td>Provides escape and unescape mechanisms for URL characters.</td>
<td>Chapter 84</td>
</tr>
<tr>
<td>ANYDATA TYPE</td>
<td>A self-describing data instance type containing an instance of the type plus a description</td>
<td>Chapter 85</td>
</tr>
<tr>
<td>ANYDATASET TYPE</td>
<td>Contains a description of a given type plus a set of data instances of that type</td>
<td>Chapter 86</td>
</tr>
<tr>
<td>ANYTYPE TYPE</td>
<td>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</td>
<td>Chapter 87</td>
</tr>
</tbody>
</table>

Note #1

The DBMS_STANDARD, STANDARD, and PLITBLM packages contain subprograms to help implement basic language features. Oracle does not recommend that the subprograms be directly called. For this reason, these three supplied packages are not documented in this book.

Note #2

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 1–7 for the subprograms provided with these packages.
**SDO_CS Package**

*Table 1–2  SDO_CS Package Subprograms*

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

**SDO_GEOM Package**

*Table 1–3  SDO_GEOM Package Subprograms*

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATE</td>
<td>Determines how two objects interact.</td>
</tr>
<tr>
<td>SDO_ARC_DENSIFY</td>
<td>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.</td>
</tr>
<tr>
<td>SDO_AREA</td>
<td>Computes the area of a two-dimensional polygon.</td>
</tr>
<tr>
<td>SDO_BUFFER</td>
<td>Generates a buffer polygon around a geometry.</td>
</tr>
<tr>
<td>SDO_CENTROID</td>
<td>Returns the centroid of a polygon.</td>
</tr>
<tr>
<td>SDO_CONVEXHULL</td>
<td>Returns a polygon-type object that represents the convex hull of a geometry object.</td>
</tr>
<tr>
<td>SDO_DIFFERENCE</td>
<td>Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_DISTANCE</td>
<td>Computes the distance between two geometry objects.</td>
</tr>
<tr>
<td>SDO_INTERSECTION</td>
<td>Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_LENGTH</td>
<td>Computes the length or perimeter of a geometry.</td>
</tr>
<tr>
<td>SDO_MAX_MBR_ORDINATE</td>
<td>Returns the maximum value for the specified ordinate of the minimum bounding rectangle of a geometry object.</td>
</tr>
<tr>
<td>SDO_MBR</td>
<td>Returns the minimum bounding rectangle of a geometry.</td>
</tr>
</tbody>
</table>
SDO_LRS Package

### Table 1-3  SDO_GEOM Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO_MIN_MBR_ORDINATE</td>
<td>Returns the minimum value for the specified ordinate of the minimum bounding rectangle of a geometry object.</td>
</tr>
<tr>
<td>SDO_POINTONSURFACE</td>
<td>Returns a point that is guaranteed to be on the surface of a polygon.</td>
</tr>
<tr>
<td>SDO_UNION</td>
<td>Returns a geometry object that is the topological union (OR operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_XOR</td>
<td>Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.</td>
</tr>
<tr>
<td>VALIDATE_GEOMETRY</td>
<td>Determines if a geometry is valid.</td>
</tr>
<tr>
<td>VALIDATE_LAYER</td>
<td>Determines if all the geometries stored in a column are valid.</td>
</tr>
<tr>
<td>WITHIN_DISTANCE</td>
<td>Determines if two geometries are within a specified Euclidean distance from one another.</td>
</tr>
</tbody>
</table>

### Table 1-4  SDO_LRS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIP_GEOM_SEGMENT</td>
<td>Clips a geometric segment (synonym of DYNAMIC_SEGMENT).</td>
</tr>
<tr>
<td>CONCATENATE_GEOM_SEGMENTS</td>
<td>Concatenates two geometric segments into one segment.</td>
</tr>
<tr>
<td>CONNECTED_GEOM_SEGMENTS</td>
<td>Checks if two geometric segments are connected.</td>
</tr>
<tr>
<td>CONVERT_TO_LRS_DIM_ARRAY</td>
<td>Converts a standard dimensional array to a Linear Referencing System dimensional array by creating a measure dimension.</td>
</tr>
<tr>
<td>CONVERT_TO_LRS_GEOM</td>
<td>Converts a standard SDO_GEOMETRY line string to a Linear Referencing System geometric segment by adding measure information.</td>
</tr>
<tr>
<td>CONVERT_TO_LRS_LAYER</td>
<td>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.</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONVERT_TO_STD_DIM_ARRAY</td>
<td>Converts a Linear Referencing System dimensional array to a standard dimensional array by removing the measure dimension.</td>
</tr>
<tr>
<td>CONVERT_TO_STD_GEOM</td>
<td>Converts a Linear Referencing System geometric segment to a standard SDO_GEOMETRY line string by removing measure information.</td>
</tr>
<tr>
<td>CONVERT_TO_STD_LAYER</td>
<td>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.</td>
</tr>
<tr>
<td>DEFINE_GEOM_SEGMENT</td>
<td>Defines a geometric segment.</td>
</tr>
<tr>
<td>DYNAMIC_SEGMENT</td>
<td>Clips a geometric segment (synonym of CLIP_GEOM_SEGMENT).</td>
</tr>
<tr>
<td>FIND_LRS_DIM_POS</td>
<td>Returns the position of the measure dimension within the SDO_DIM_ARRAY structure for a specified SDO_GEOMETRY column.</td>
</tr>
<tr>
<td>FIND_MEASURE</td>
<td>Returns the measure of the closest point on a segment to a specified projection point.</td>
</tr>
<tr>
<td>GEOM_SEGMENT_END_MEASURE</td>
<td>Returns the end measure of a geometric segment.</td>
</tr>
<tr>
<td>GEOM_SEGMENT_END_PT</td>
<td>Returns the end point of a geometric segment.</td>
</tr>
<tr>
<td>GEOM_SEGMENT_LENGTH</td>
<td>Returns the length of a geometric segment.</td>
</tr>
<tr>
<td>GEOM_SEGMENT_START_MEASURE</td>
<td>Returns the start measure of a geometric segment.</td>
</tr>
<tr>
<td>GEOM_SEGMENT_START_PT</td>
<td>Returns the start point of a geometric segment.</td>
</tr>
<tr>
<td>GET_MEASURE</td>
<td>Returns the measure of an LRS point.</td>
</tr>
<tr>
<td>IS_GEOM_SEGMENT_DEFINED</td>
<td>Checks if an LRS segment is defined correctly.</td>
</tr>
<tr>
<td>IS_MEASURE_DECREASING</td>
<td>Checks if the measure values along an LRS segment are decreasing (that is, descending in numerical value).</td>
</tr>
</tbody>
</table>
### Table 1–4 SDO_LRS Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_MEASURE_INCREASING</td>
<td>Checks if the measure values along an LRS segment are increasing (that is, ascending in numerical value).</td>
</tr>
<tr>
<td>LOCATE_PT</td>
<td>Returns the point located at a specified distance from the start of a geometric segment.</td>
</tr>
<tr>
<td>MEASURE_RANGE</td>
<td>Returns the measure range of a geometric segment, that is, the difference between the start measure and end measure.</td>
</tr>
<tr>
<td>MEASURE_TO_PERCENTAGE</td>
<td>Returns the percentage (0 to 100) that a specified measure is of the measure range of a geometric segment.</td>
</tr>
<tr>
<td>OFFSET_GEOM_SEGMENT</td>
<td>Returns the geometric segment at a specified offset from a geometric segment.</td>
</tr>
<tr>
<td>PERCENTAGE_TO_MEASURE</td>
<td>Returns the measure value of a specified percentage (0 to 100) of the measure range of a geometric segment.</td>
</tr>
<tr>
<td>PROJECT_PT</td>
<td>Returns the projection point of a point on a geometric segment.</td>
</tr>
<tr>
<td>REDEFINE_GEOM_SEGMENT</td>
<td>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.</td>
</tr>
<tr>
<td>RESET_MEASURE</td>
<td>Sets all measures of a geometric segment, including the start and end measures, to null values, overriding any previously assigned measures.</td>
</tr>
<tr>
<td>REVERSE_MEASURE</td>
<td>Returns a new geometric segment by reversing the original geometric segment.</td>
</tr>
<tr>
<td>REVERSE_GEOMETRY</td>
<td>Returns a new geometric segment by reversing the measure values and the direction of the original geometric segment.</td>
</tr>
<tr>
<td>SCALE_GEOM_SEGMENT</td>
<td>Scales a geometric segment.</td>
</tr>
<tr>
<td>SET_PT_MEASURE</td>
<td>Sets the measure value of a specified point.</td>
</tr>
<tr>
<td>SPLIT_GEOM_SEGMENT</td>
<td>Splits a geometric segment into two segments.</td>
</tr>
</tbody>
</table>
Table 1–5  SDO_MIGRATE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM_815_TO_81X</td>
<td>Migrates data from Spatial release 8.1.5 to the current release.</td>
</tr>
<tr>
<td>OGIS_METADATA_FROM</td>
<td>Generates a temporary table used when migrating OGIS (OpenGIS) metadata tables.</td>
</tr>
<tr>
<td>OGIS_METADATA_TO</td>
<td>Reads a temporary table used when migrating OGIS metadata tables.</td>
</tr>
<tr>
<td>TO_734</td>
<td>Migrates data from a previous release of Spatial Data Option to release 7.3.4.</td>
</tr>
<tr>
<td>TO_81X</td>
<td>Migrates tables from Spatial Data Option 7.3.4 or Spatial Cartridge 8.0.4 to Oracle Spatial.</td>
</tr>
<tr>
<td>TO_CURRENT</td>
<td>Migrates data from a previous Spatial release to the current release.</td>
</tr>
</tbody>
</table>
# SDO_TUNE Package

## Table 1–6 SDO_TUNE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE RTREE</td>
<td>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.</td>
</tr>
<tr>
<td>AVERAGE MBR</td>
<td>Calculates the average minimum bounding rectangle for geometries in a layer.</td>
</tr>
<tr>
<td>ESTIMATE_INDEX_ PERFORMANCE</td>
<td>Estimates the spatial index selectivity.</td>
</tr>
<tr>
<td>ESTIMATE_TILING_LEVEL</td>
<td>Determines an appropriate tiling level for creating fixed-size index tiles.</td>
</tr>
<tr>
<td>ESTIMATE_TILING_TIME</td>
<td>Estimates the tiling time for a layer, in seconds.</td>
</tr>
<tr>
<td>ESTIMATE_TOTAL_NUMTILES</td>
<td>Estimates the total number of spatial tiles for a layer.</td>
</tr>
<tr>
<td>EXTENT_OF</td>
<td>Determines the minimum bounding rectangle of the data in a layer.</td>
</tr>
<tr>
<td>HISTOGRAM_ANALYSIS</td>
<td>Calculates statistical histograms for a spatial layer.</td>
</tr>
<tr>
<td>MIX_INFO</td>
<td>Calculates geometry type information for a spatial layer, such as the percentage of each geometry type.</td>
</tr>
<tr>
<td>QUALITY_DEGRADATION</td>
<td>Returns the quality degradation for an R-tree index or the average quality degradation for all index tables for an R-tree index.</td>
</tr>
<tr>
<td>RTREE_QUALITY</td>
<td>Returns the quality score for an R-tree index or the average quality score for all index tables for an R-tree index.</td>
</tr>
</tbody>
</table>

# UTL_PG Package

## Table 1–7 UTL_PG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE NUMBER TO RAW FORMAT</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 1–7  UTL_PG Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE_RAW_TO_NUMBER_</td>
<td>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.</td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
</tr>
<tr>
<td>NUMBER_TO_RAW</td>
<td>Converts an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format.</td>
</tr>
<tr>
<td>NUMBER_TO_RAW_</td>
<td>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.</td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
</tr>
<tr>
<td>RAW_TO_NUMBER</td>
<td>Converts a RAW byte-string from the remote host internal format into an Oracle number.</td>
</tr>
<tr>
<td>RAW_TO_NUMBER_</td>
<td>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.</td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
</tr>
<tr>
<td>WMSG</td>
<td>Extracts a warning message specified by wmsgitem from wmsgblk.</td>
</tr>
<tr>
<td>WMSGCNT</td>
<td>Tests a wmsgblk to determine how many warnings, if any, are present.</td>
</tr>
</tbody>
</table>
DBMS_ALERT supports asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can cause itself to be notified 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 does not get alerted until the transaction signalling the alert commits. There can be any number of concurrent signallers 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. This table shows the messages and the procedures that can raise them.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORU-10001 lock request error, status: N</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10015 error: N waiting for pipe status</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10016 error: N sending on pipe ’X’</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10017 error: N receiving on pipe ’X’</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10019 error: N on lock request</td>
<td>WAIT</td>
</tr>
<tr>
<td>ORU-10020 error: N on lock request</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10021 lock request error; status: N</td>
<td>REGISTER</td>
</tr>
<tr>
<td>ORU-10022 lock request error, status: N</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10023 lock request error; status N</td>
<td>WAITONE</td>
</tr>
<tr>
<td>ORU-10024 there are no alerts registered</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10025 lock request error; status N</td>
<td>REGISTER</td>
</tr>
<tr>
<td>ORU-10037 attempting to wait on uncommitted signal from same session</td>
<td>WAITONE</td>
</tr>
</tbody>
</table>
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 41, "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

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

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

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

Parameters

Table 2–3  REGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert in which this session is interested.</td>
</tr>
</tbody>
</table>
Summary of DBMS_ALERT Subprograms

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert (case-insensitive) to be removed from registration list.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitivity</td>
<td>Polling interval, in seconds, to sleep between polls. The default interval is five seconds.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert to signal.</td>
</tr>
<tr>
<td>message</td>
<td>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.</td>
</tr>
</tbody>
</table>

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);  
```
WAITONE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Returns the name of the alert that occurred.</td>
</tr>
<tr>
<td>message</td>
<td>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.</td>
</tr>
<tr>
<td>status</td>
<td>Values returned: 0 - alert occurred 1 - time-out occurred</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum time to wait for an alert. If no alert occurs before timeout seconds, this returns a status of 1.</td>
</tr>
</tbody>
</table>

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

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

Table 2–8  WAITONE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert to wait for.</td>
</tr>
<tr>
<td>message</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 above example guarantees that the application always sees the latest data,
although it may not see every intermediate value.
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, then 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:

- Privileges
- Summary of `DBMS_APPLICATION_INFO` Subprograms
Privileges

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

Summary of DBMS_APPLICATION_INFO Subprograms

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

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

Table 3–2 SET_MODULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_name</td>
<td>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.</td>
</tr>
<tr>
<td>action_name</td>
<td>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.</td>
</tr>
</tbody>
</table>

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);
READ_MODULE Procedure

Parameters

Table 3–3  SET_ACTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_name</td>
<td>The name of the current action within the current module.</td>
</tr>
<tr>
<td></td>
<td>When the current action terminates, call this procedure with the name of</td>
</tr>
<tr>
<td></td>
<td>the next action if there is one, or NULL if there is not. Names longer than</td>
</tr>
<tr>
<td></td>
<td>32 bytes are truncated.</td>
</tr>
</tbody>
</table>

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, then 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:

```sql
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

```sql
DBMS_APPLICATION_INFO.READ_MODULE (module_name OUT VARCHAR2,)
```
action_name OUT VARCHAR2);

### Parameters

**Table 3–4 READ_MODULE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_name</td>
<td>Last value that the module name was set to by calling SET_MODULE.</td>
</tr>
<tr>
<td>action_name</td>
<td>Last value that the action name was set to by calling SET_ACTION or SET_MODULE.</td>
</tr>
</tbody>
</table>

### 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.

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

<table>
<thead>
<tr>
<th>SQL_TEXT DISK_READS MODULE ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO emp_1 add_employee insert into emp</td>
</tr>
<tr>
<td>(ename, empno, sal, mgr, job, hiredate, comm, deptno)</td>
</tr>
<tr>
<td>VALUES</td>
</tr>
<tr>
<td>(name, next.emp_seq, manager, title, SYSDATE, commission, department)</td>
</tr>
</tbody>
</table>

1 row selected.

### SET_CLIENT_INFO Procedure

This procedure supplies additional information about the client application.

### Syntax

```sql
DBMS_APPLICATION_INFO.SET_CLIENT_INFO ( 
```
client_info IN VARCHAR2);

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_info</td>
<td>Last client information value supplied to the SET_CLIENT_INFO procedure.</td>
</tr>
</tbody>
</table>

SET_SESSION_LONGOPS Procedure

This procedure sets a row in the $SESSION_LONGOPS view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.
Applications may use the `set_session_longops` procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the `V$SESSION_LONGOPS` view.

**Syntax**

```sql
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)
```

```sql
set_session_longops_nohint constant BINARY_INTEGER := -1;
```

**Pragmas**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rindex</td>
<td>A token which represents the <code>v$session_longops</code> row to update. Set this to <code>set_session_longops_nohint</code> to start a new row. Use the returned value from the prior call to reuse a row.</td>
</tr>
<tr>
<td>slno</td>
<td>Saves information across calls to <code>set_session_longops</code>: It is for internal use and should not be modified by the caller.</td>
</tr>
<tr>
<td>op_name</td>
<td>Specifies the name of the long running task. It appears as the <code>OPNAME</code> column of <code>v$session_longops</code>. The maximum length is 64 bytes.</td>
</tr>
<tr>
<td>target</td>
<td>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 <code>TARGET</code> column of <code>v$session_longops</code>.</td>
</tr>
<tr>
<td>context</td>
<td>Any number the client wants to store. It appears in the <code>CONTEXT</code> column of <code>v$session_longops</code>.</td>
</tr>
<tr>
<td>sofar</td>
<td>Any number the client wants to store. It appears in the <code>SO FAR</code> column of <code>v$session_longops</code>. This is typically the amount of work which has been done so far.</td>
</tr>
<tr>
<td>totalwork</td>
<td>Any number the client wants to store. It appears in the <code>TOTAL WORK</code> column of <code>v$session_longops</code>. This is typically an estimate of the total amount of work needed to be done in this long running operation.</td>
</tr>
<tr>
<td>target_desc</td>
<td>Specifies the description of the object being manipulated in this long operation. This provides a caption for the <code>target</code> parameter. This value appears in the <code>TARGET_DESC</code> field of <code>v$session_longops</code>. The maximum length is 32 bytes.</td>
</tr>
<tr>
<td>units</td>
<td>Specifies the units in which <code>sofar</code> and <code>totalwork</code> are being represented. It appears as the <code>UNITS</code> field of <code>v$session_longops</code>. The maximum length is 32 bytes.</td>
</tr>
</tbody>
</table>
Example

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

```sql
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;
```
SET_SESSION_LONGOPS Procedure
The DBMS_AQ package provides an interface to Oracle’s Advanced Queuing.

See Also: Oracle9i Application Developer’s Guide - Advanced Queuing contains information about using DBMS_AQ.
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>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>IMMEDIATE, ON_COMMIT</td>
</tr>
<tr>
<td>dequeue mode</td>
<td>BROWSE, LOCKED, REMOVE, REMOVE_NODATA</td>
</tr>
<tr>
<td>navigation</td>
<td>FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION</td>
</tr>
<tr>
<td>state</td>
<td>WAITING, READY, PROCESSED, EXPIRED</td>
</tr>
<tr>
<td>sequence_deviation</td>
<td>BEFORE, TOP</td>
</tr>
<tr>
<td>wait</td>
<td>FOREVER, NO_WAIT</td>
</tr>
<tr>
<td>delay</td>
<td>NO_DELAY</td>
</tr>
<tr>
<td>expiration</td>
<td>NEVER</td>
</tr>
<tr>
<td>namespace</td>
<td>NAMESPACE_AQ, NAMESPACE_ANONYMOUS</td>
</tr>
</tbody>
</table>

**Data Structures for DBMS_AQ**

Table 4-2 lists the data structures used in both `DBMS_AQ` and `DBMS_AQADM`.

<table>
<thead>
<tr>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Object Name&quot; on page 4-3</td>
</tr>
<tr>
<td>&quot;Type Name&quot; on page 4-3</td>
</tr>
<tr>
<td>&quot;Agent&quot; on page 4-4</td>
</tr>
<tr>
<td>&quot;Enqueue Options Type&quot; on page 4-5</td>
</tr>
</tbody>
</table>
Object Name

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

Syntax

```sql
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, then 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.
Agent

Syntax

type_name := VARCHAR2;

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;object_types&gt;</td>
<td>Maximum number of attributes in the object type is limited to 900.</td>
</tr>
<tr>
<td>&quot;RAW&quot;</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

Agent

The aq$_agent data structure identifies a producer or a consumer of a message.

Syntax

```plsql
TYPE sys.aq$_agent IS OBJECT (  
    name   VARCHAR2(30),
    address VARCHAR2(1024),
    protocol NUMBER);
```
Attributes

Table 4–4 Agent Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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.</td>
</tr>
<tr>
<td>address</td>
<td>Protocol-specific address of the recipient. If the protocol is 0 (default), then the address is of the form [schema.]queue[@dblink].</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol to interpret the address and propagate the message. The default is 0.</td>
</tr>
</tbody>
</table>

Enqueue Options Type

The enqueue_options_t data structure specifies the options available for the enqueue operation.

Syntax

```sql
TYPE 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 4–5 Enqueue Options Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>Specifies the transactional behavior of the enqueue request. ON_COMMIT: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This is the default case. 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.</td>
</tr>
</tbody>
</table>
Dequeue Options Type

The `dequeue_options_t` data structure specifies the options available for the dequeue operation.

### Syntax

```plsql
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);
```

### Table 4–5 Enqueue Options Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relative_msgid</td>
<td>Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid if, and only if, BEFORE is specified in <code>sequence_deviation</code>. This parameter is ignored if sequence deviation is not specified.</td>
</tr>
<tr>
<td>sequence_deviation</td>
<td>Specifies whether the message being enqueued should be dequeued before other message(s) already in the queue.</td>
</tr>
<tr>
<td>BEFORE</td>
<td>The message is enqueued ahead of the message specified by <code>relative_msgid</code>.</td>
</tr>
<tr>
<td>TOP</td>
<td>The message is enqueued ahead of any other messages.</td>
</tr>
<tr>
<td>NULL</td>
<td>Default</td>
</tr>
<tr>
<td>transformation</td>
<td>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.</td>
</tr>
</tbody>
</table>

Dequeue Options Type

The `dequeue_options_t` data structure specifies the options available for the dequeue operation.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_name</td>
<td>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.</td>
</tr>
<tr>
<td>dequeue_mode</td>
<td>Specifies the locking behavior associated with the dequeue. - <strong>BROWSE</strong>: Read the message without acquiring any lock on the message. This is equivalent to a select statement. - <strong>LOCKED</strong>: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This is equivalent to a select for update statement. - <strong>REMOVE</strong>: Read the message and update or delete it. This is the default. The message can be retained in the queue table based on the retention properties. - <strong>REMOVE_NODATA</strong>: Mark the message as updated or deleted. The message can be retained in the queue table based on the retention properties.</td>
</tr>
<tr>
<td>navigation</td>
<td>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. - <strong>NEXT_MESSAGE</strong>: Retrieve the next message which is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message which matches the search criteria and belongs to the message group. This is the default. - <strong>NEXT_TRANSACTION</strong>: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue. - <strong>FIRST_MESSAGE</strong>: Retrieves the first message which is available and matches the search criteria. This resets the position to the beginning of the queue.</td>
</tr>
</tbody>
</table>
Dequeue Options Type

Table 4-6  Dequeue Options Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>Specifies whether the new message is dequeued as part of the current transaction. The visibility parameter is ignored when using the BROWSE mode.</td>
</tr>
<tr>
<td></td>
<td><strong>ON_COMMIT</strong>: The dequeue will be part of the current transaction. This is the default case.</td>
</tr>
<tr>
<td></td>
<td><strong>IMMEDIATE</strong>: The dequeued message is not part of the current transaction. It constitutes a transaction on its own.</td>
</tr>
<tr>
<td>wait</td>
<td>Specifies the wait time if there is currently no message available which matches the search criteria.</td>
</tr>
<tr>
<td></td>
<td><strong>FOREVER</strong>: wait forever. This is the default.</td>
</tr>
<tr>
<td></td>
<td><strong>NO_WAIT</strong>: do not wait</td>
</tr>
<tr>
<td></td>
<td><strong>number</strong>: wait time in seconds</td>
</tr>
<tr>
<td>msgid</td>
<td>Specifies the message identifier of the message to be dequeued.</td>
</tr>
<tr>
<td>correlation</td>
<td>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.</td>
</tr>
<tr>
<td>deq_condition</td>
<td>A conditional expression based on the message properties, the message data properties and PL/SQL functions.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.</td>
</tr>
</tbody>
</table>
Message Properties Type

The message_properties_t data structure describes the information that is used by AQ to manage individual messages. These are set at enqueue time, and their values are returned at dequeue time.

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);
```

```
TYPE aq$_recipient_list_t IS TABLE OF sys.aq$_agent  
    INDEX BY BINARY_INTEGER;
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Specifies/returns the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.</td>
</tr>
<tr>
<td>delay</td>
<td>Specifies/returns 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 will be in the WAITING state, when the delay expires the messages goes to the READY state. DELAY processing requires the queue monitor to be started. Note that delay is set by the producer who enqueues the message.</td>
</tr>
</tbody>
</table>

`NO_DELAY`: the message is available for immediate dequeuing.  
`number`: the number of seconds to delay the message.
Message Properties Type

Table 4–7  Message Properties Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| expiration     | Specifies/returns 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.  
NEVER: message does not expire.  
number: 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 this message. This parameter cannot be set at enqueue time.                     |
| recipient_list | For type definition, see the "Agent" on page 4-4.                                                                                             |
| This parameter is only valid for queues which allow multiple consumers. The default recipients are the queue subscribers.  
This parameter is not returned to a consumer at dequeue time. |
| exception_queue| Specifies/returns the name of the queue to which the message is moved if it cannot be processed successfully. Messages are moved in two cases: The number of unsuccessful dequeue attempts has exceeded max_retries or the message has expired. 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 used, then the parameter returns a NULL value at dequeue time. |
| enqueue_time   | Returns the time the message was enqueued. This value is determined by the system and cannot be set by the user. This parameter can not be set at enqueue time. |
**AQ Recipient List Type**

The `aq$_recipient_list_t` data structure identifies the list of agents that will receive the message. It is used only when the queue is enabled for multiple dequeues.

**Syntax**

```plaintext
TYPE aq$_recipient_list_t IS TABLE OF sys.aq$_agent
   INDEX BY BINARY_INTEGER;
```

**AQ Agent List Type**

The `aq$_agent_list_t` data structure identifies the list of agents for `DBMS_AQ.LISTEN` to listen for.

**Syntax**

```plaintext
TYPE aq$_agent_list_t IS TABLE of sys.aq$_agent
   INDEX BY BINARY_INTEGER;
```

**AQ Subscriber List Type**

The `aq$_subscriber_list_t` data structure identifies the list of subscribers that subscribe to this queue.

---

**Table 4–7  Message Properties Type Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>state</code></td>
<td>Returns the state of the message at the time of the dequeue.</td>
</tr>
<tr>
<td></td>
<td>This parameter can not be set at enqueue time.</td>
</tr>
<tr>
<td></td>
<td>0: The message is ready to be processed.</td>
</tr>
<tr>
<td></td>
<td>1: The message delay has not yet been reached.</td>
</tr>
<tr>
<td></td>
<td>2: The message has been processed and is retained.</td>
</tr>
<tr>
<td></td>
<td>3: The message has been moved to the exception queue.</td>
</tr>
<tr>
<td><code>sender_id</code></td>
<td>Specifies/returns the application-specified sender identification.</td>
</tr>
<tr>
<td></td>
<td>DEFAULT: NULL</td>
</tr>
<tr>
<td><code>original_msgid</code></td>
<td>This parameter is used by Oracle AQ for propagating messages.</td>
</tr>
<tr>
<td></td>
<td>DEFAULT: NULL</td>
</tr>
</tbody>
</table>

---

**state** Returns the state of the message at the time of the dequeue. This parameter can not be set at enqueue time.

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/returns the application-specified sender identification.

**DEFAULT:** NULL

**original_msgid** This parameter is used by Oracle AQ for propagating messages.

**DEFAULT:** NULL
AQ Registration Info Type

Syntax

```
TYPE aq$_subscriber_list_t IS TABLE OF sys.aq$_agent
    INDEX BY BINARY_INTEGER;
```

AQ Registration Info Type

The `aq$_reg_info` data structure identifies a producer or a consumer of a message.

Syntax

```
TYPE sys.aq$_reg_info IS OBJECT (
    name      VARCHAR2(128),
    namespace NUMBER,
    callback  VARCHAR2(4000),
    context   RAW(2000));
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the name of the subscription. The subscription name is of the form <code>&lt;schema&gt;.&lt;queue&gt;</code> if the registration is for a single consumer queue and <code>&lt;schema&gt;.&lt;queue&gt;:&lt;consumer_name&gt;</code> if the registration is for a multiconsumer queues.</td>
</tr>
<tr>
<td>namespace</td>
<td>Specifies the namespace of the subscription. To receive notifications from AQ queues the namespace must be <code>DBMS_AQ.NAMESPACE_AQ</code>. To receive notifications from other applications via <code>DBMS_AQ.POST</code> or <code>OCISubscriptionPost()</code>, the namespace must be <code>DBMS_AQ.NAMESPACE_ANONYMOUS</code>.</td>
</tr>
<tr>
<td>callback</td>
<td>For HTTP notifications, the form is <code>http://www.company.com:8080</code></td>
</tr>
</tbody>
</table>
Table 4–9 shows the actions performed when different notification mechanisms/presentations are specified for nonpersistent queues.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>Specifies the action to be performed on message notification. For email notifications, the form is mailto://xyz@company.com For AQ PL/SQL Callback, use plsql://&lt;schema&gt;.&lt;procedure&gt;?PR=0 for raw message payload OR plsql://&lt;schema&gt;.&lt;procedure&gt;?PR=1 for ADT message payload converted to XML</td>
</tr>
<tr>
<td>context</td>
<td>Specifies the context that is to be passed to the callback function. Default: NULL</td>
</tr>
</tbody>
</table>

Table 4–9  Nonpersistent Queues

<table>
<thead>
<tr>
<th>Queue Payload Type</th>
<th>RAW TYPE</th>
<th>XML TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Presentation Specified</td>
<td>Notification Mechanism</td>
</tr>
<tr>
<td></td>
<td>RAW</td>
<td>XML</td>
</tr>
<tr>
<td></td>
<td>OCI</td>
<td>Email</td>
</tr>
<tr>
<td>RAW</td>
<td>The callback receives the RAW data in the payload.</td>
<td>Not supported</td>
</tr>
<tr>
<td>ADT</td>
<td>Not supported.</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>
AQ Registration Info List Type

The `aq$_reg_info_list` data structure identifies the list of registrations to a queue.

Syntax

```plaintext
TYPE aq$_reg_info_list AS VARRAY(1024) OF sys.aq$_reg_info
```

AQ Notification Descriptor Type

The `aq$_descriptor` data structure specifies the AQ Descriptor received by the AQ PL/SQL callbacks upon notification. See "AQ PL/SQL Callback" on page 4-15.

Syntax

```plaintext
TYPE sys.aq$_descriptor IS OBJECT (  
    queue_name VARCHAR2(30),  
    consumer_name VARCHAR2(30),  
    msg_id RAW(16),  
    msg_prop msg_prop_t);
```

Attributes

<table>
<thead>
<tr>
<th>Table 4–10 AQ Notification Descriptor Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute</strong></td>
</tr>
<tr>
<td>queue_name</td>
</tr>
<tr>
<td>consumer_name</td>
</tr>
<tr>
<td>msg_id</td>
</tr>
<tr>
<td>msg_prop</td>
</tr>
</tbody>
</table>

AQ Post Info Type

The `aq$_post_info` data structure specifies anonymous subscriptions to which you want to post messages.

Syntax

```plaintext
TYPE sys.aq$_post_info IS OBJECT (  
    name VARCHAR2(128),
```

4-14 Supplied PL/SQL Packages and Types Reference
namespace NUMBER,
payload RAW(2000));

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the anonymous subscription to which you want to post to.</td>
</tr>
<tr>
<td>namespace</td>
<td>To receive notifications from other applications via DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.</td>
</tr>
<tr>
<td>payload</td>
<td>The payload to be posted to the anonymous subscription. Default: NULL.</td>
</tr>
</tbody>
</table>

**AQ Post Info List Type**

The aq$_post_info_list data structure identifies the list of anonymous subscriptions to which you want to post messages.

**Syntax**

```plsql
type aq$_post_info_list as varray(1024) of sys.aq$_post_info
```

**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 an raw payload enqueue then this PL/SQL callback must have the following signature:

```plsql
procedure plsqlcallback(
    context out raw,
    reginfo out sys.aq$_reg_info,
    descr out sys.aq$_descriptor,
    payload out raw,
    payloadl out number);
```
Attributes

Table 4–12 AQ PL/SQL Callback Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Specifies the context for the callback function that was passed by <code>dbms_aq.register</code>. See &quot;AQ Registration Info Type&quot; on page 4-12.</td>
</tr>
<tr>
<td>reginfo</td>
<td>See &quot;AQ Registration Info Type&quot; on page 4-12.</td>
</tr>
<tr>
<td>descr</td>
<td>See &quot;AQ Notification Descriptor Type&quot; on page 4-14.</td>
</tr>
<tr>
<td>payload</td>
<td>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.</td>
</tr>
<tr>
<td>payloadl</td>
<td>Specifies the length of <code>payload</code>. If <code>payload</code> is null, <code>payloadl = 0</code>.</td>
</tr>
</tbody>
</table>

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

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

Summary of Subprograms

Table 4–13 DBMS_AQ Package Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQUEUE Procedure on page 4-17</td>
<td>Adds a message to the specified queue.</td>
</tr>
<tr>
<td>DEQUEUE Procedure on page 4-19</td>
<td>Dequeues a message from the specified queue.</td>
</tr>
<tr>
<td>LISTEN Procedure on page 4-21</td>
<td>Listen to one or more queues on behalf of a list of agents.</td>
</tr>
</tbody>
</table>
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);
```

### Table 4–13 DBMS_AQ Package Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER Procedure on page 4–22</td>
<td>Register for message notifications</td>
</tr>
<tr>
<td>UNREGISTER Procedure on page 4–23</td>
<td>Unregister a subscription which turns off notification</td>
</tr>
<tr>
<td>POST Procedure on page 4–24</td>
<td>Posts to a anonymous subscription which allows all clients who are registered for the subscription to get notifications.</td>
</tr>
<tr>
<td>BIND_AGENT Procedure on page 4–25</td>
<td>Creates an entry for an AQ agent in the LDAP directory</td>
</tr>
<tr>
<td>UNBIND_AGENT Procedure on page 4–25</td>
<td>Removes an entry for an AQ agent from the LDAP directory</td>
</tr>
</tbody>
</table>

**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.
Parameters

Table 4–14  ENQUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.</td>
</tr>
<tr>
<td>enqueue_options</td>
<td>See “Enqueue Options Type” on page 4-5.</td>
</tr>
<tr>
<td>message_properties</td>
<td>See “Message Properties Type” on page 4-9.</td>
</tr>
<tr>
<td>payload</td>
<td>Not interpreted by Oracle AQ.</td>
</tr>
<tr>
<td></td>
<td>The payload must be specified according to the specification in the associated queue table. NULL is an acceptable parameter.</td>
</tr>
<tr>
<td></td>
<td>For the definition of &lt;type_name&gt; please refer to “Type Name” on page 4-3.</td>
</tr>
<tr>
<td>msgid</td>
<td>System generated identification of the message.</td>
</tr>
<tr>
<td></td>
<td>This is a globally unique identifier that can be used to identify the message at dequeue time.</td>
</tr>
</tbody>
</table>

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.
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of the queue.</td>
</tr>
<tr>
<td>dequeue_options</td>
<td>See &quot;Dequeue Options Type&quot; on page 4-6.</td>
</tr>
<tr>
<td>message_properties</td>
<td>See &quot;Message Properties Type&quot; on page 4-9.</td>
</tr>
<tr>
<td>payload</td>
<td>Not interpreted by Oracle AQ. The payload must be specified according to the specification in the associated queue table. For the definition of <code>&lt;type_name&gt;</code> please refer to &quot;Type Name&quot; on page 4-3.</td>
</tr>
<tr>
<td>msgid</td>
<td>System generated identification of the message.</td>
</tr>
</tbody>
</table>

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...
DEQUEUE Procedure

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 \texttt{tab.user\_data} as a qualifier to indicate the specific column of the queue table that stores the payload.

Example: \texttt{tab.user\_data\_orderstatus='EXPRESS'}

Only messages in the \texttt{READY} state are dequeued unless \texttt{msgid} is specified.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the \texttt{msgid} and correlation ID in \texttt{dequeue\_options}.

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

The default \texttt{NAVIGATION} parameter during dequeue is \texttt{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 \texttt{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.

\textbf{Note}: It may be more efficient to use the \texttt{FIRST\_MESSAGE} navigation option when messages are concurrently enqueued. If the \texttt{FIRST\_MESSAGE} option is not specified, AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the \texttt{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.

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_list</td>
<td>List of agents to listen for.</td>
</tr>
</tbody>
</table>

REGISTER Procedure

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, then only the first agent listed is returned. If there are no messages found when the wait time expires, then 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 non-persistent queues.

REGISTER Procedure

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

Syntax

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

Table 4-17  REGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_list</td>
<td>Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ Registration Info Type.</td>
</tr>
<tr>
<td>count</td>
<td>Specifies the number of entries in the reg_list.</td>
</tr>
</tbody>
</table>

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, then 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 6, "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 6, "DBMS_AQELM" for more information on HTTP notifications.

UNREGISTER Procedure

This procedure unregisters a subscription which turns off notifications.

Syntax

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

Parameters

Table 4–18  UNREGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_list</td>
<td>Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ Registration Info Type.</td>
</tr>
<tr>
<td>count</td>
<td>Specifies the number of entries in the reg_list.</td>
</tr>
</tbody>
</table>

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 which 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 4–19  POST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_list</td>
<td>Specifies the list of anonymous subscriptions to which you want to post. It is a list of AQ Post Info Type.</td>
</tr>
<tr>
<td>count</td>
<td>Specifies the number of entries in the post_list.</td>
</tr>
</tbody>
</table>

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>Agent that is to be registered in LDAP server</td>
</tr>
<tr>
<td>certificate</td>
<td>Location (LDAP distinguished name) of the &quot;organizationalperson&quot; entry in LDAP whose digital certificate (attribute usercertificate) is to be used for this agent. Example: &quot;cn=OE, cn=ACME, cn=com&quot; is a DN for a OrganizationalPerson OE whose certificate will be used with the agent specified above</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent:</td>
<td>Agent that is to be removed from the LDAP server</td>
</tr>
</tbody>
</table>
The `DBMS_AQADM` package provides procedures to manage Advanced Queuing configuration and administration information.

**See Also:** *Oracle9i Application Developer’s Guide - Advanced Queuing* contains information about using `DBMS_AQADM`.

This chapter discusses the following topics:

- **Enumerated Constants**
- **Summary of DBMS_AQADM Subprograms**
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>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention</td>
<td>0, 1, 2 ... INFINITE</td>
</tr>
<tr>
<td>message_grouping</td>
<td>TRANSACTIONAL, NONE</td>
</tr>
<tr>
<td>queue_type</td>
<td>NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE</td>
</tr>
</tbody>
</table>

See Also: For more information on the Java classes and data structures used in both DBMS_AQ and DBMS_AQADM, see Chapter 4, "DBMS_AQ"

Summary of DBMS_AQADM Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CREATE_QUEUE_TABLE Procedure&quot; on page 5-4</td>
<td>Creates a queue table for messages of a predefined type.</td>
</tr>
<tr>
<td>&quot;ALTER_QUEUE_TABLE Procedure&quot; on page 5-8</td>
<td>Alters an existing queue table.</td>
</tr>
<tr>
<td>&quot;DROP_QUEUE_TABLE Procedure&quot; on page 5-9</td>
<td>Drops an existing queue table.</td>
</tr>
<tr>
<td>&quot;CREATE_QUEUE Procedure&quot; on page 5-9</td>
<td>Creates a queue in the specified queue table.</td>
</tr>
<tr>
<td>&quot;CREATE_NP_QUEUE Procedure&quot; on page 5-11</td>
<td>Creates a nonpersistent RAW queue.</td>
</tr>
<tr>
<td>&quot;ALTER_QUEUE Procedure&quot; on page 5-12</td>
<td>Alters existing properties of a queue.</td>
</tr>
<tr>
<td>&quot;DROP_QUEUE Procedure&quot; on page 5-14</td>
<td>Drops an existing queue.</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;START_QUEUE Procedure&quot; on page 5-14</td>
<td>Enables the specified queue for enqueuing and/or dequeuing.</td>
</tr>
<tr>
<td>&quot;STOP_QUEUE Procedure&quot; on page 5-15</td>
<td>Disables enqueuing and/or dequeuing on the specified queue.</td>
</tr>
<tr>
<td>&quot;GRANT_SYSTEM_PRIVILEGE Procedure&quot; on page 5-16</td>
<td>Grants AQ system privileges to users and roles.</td>
</tr>
<tr>
<td>&quot;REVOKE_SYSTEM_PRIVILEGE Procedure&quot; on page 5-17</td>
<td>Revokes AQ system privileges from users and roles.</td>
</tr>
<tr>
<td>&quot;GRANT_QUEUE_PRIVILEGE Procedure&quot; on page 5-18</td>
<td>Grants privileges on a queue to users and roles.</td>
</tr>
<tr>
<td>&quot;REVOKE_QUEUE_PRIVILEGE Procedure&quot; on page 5-19</td>
<td>Revokes privileges on a queue from users and roles.</td>
</tr>
<tr>
<td>&quot;ADD_SUBSCRIBER Procedure&quot; on page 5-19</td>
<td>Adds a default subscriber to a queue.</td>
</tr>
<tr>
<td>&quot;ALTER_SUBSCRIBER Procedure&quot; on page 5-21</td>
<td>Alters existing properties of a subscriber to a specified queue.</td>
</tr>
<tr>
<td>&quot;REMOVE_SUBSCRIBER Procedure&quot; on page 5-21</td>
<td>Removes a default subscriber from a queue.</td>
</tr>
<tr>
<td>&quot;SCHEDULE_PROPAGATION Procedure&quot; on page 5-22</td>
<td>Schedules propagation of messages from a queue to a destination identified by a specific dblink.</td>
</tr>
<tr>
<td>&quot;UNSCHEDULE_PROPAGATION Procedure&quot; on page 5-23</td>
<td>Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific dblink.</td>
</tr>
<tr>
<td>&quot;VERIFY_QUEUE_TYPES Procedure&quot; on page 5-24</td>
<td>Verifies that the source and destination queues have identical types.</td>
</tr>
<tr>
<td>&quot;ALTER_PROPAGATION_SCHEDULE Procedure&quot; on page 5-25</td>
<td>Alters parameters for a propagation schedule.</td>
</tr>
<tr>
<td>&quot;ENABLE_PROPAGATION_SCHEDULE Procedure&quot; on page 5-27</td>
<td>Enables a previously disabled propagation schedule.</td>
</tr>
<tr>
<td>&quot;DISABLE_PROPAGATION_SCHEDULE Procedure&quot; on page 5-27</td>
<td>Disables a propagation schedule.</td>
</tr>
</tbody>
</table>
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 Oracle8i-compatible queue tables, the following two index organized tables are created:

---

Table 5–2  DBMS_AQADM Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;MIGRATE_QUEUE_TABLE Procedure&quot; on page 5-28</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;CREATE_AQ_AGENT Procedure&quot; on page 5-28</td>
<td>Registers a agent for AQ Internet access</td>
</tr>
<tr>
<td>&quot;ALTER_AQ_AGENT Procedure&quot; on page 5-29</td>
<td>Alters a agent registered for AQ Internet access</td>
</tr>
<tr>
<td>&quot;DROP_AQ_AGENT Procedure&quot; on page 5-30</td>
<td>Drops a agent registered for AQ Internet access</td>
</tr>
<tr>
<td>&quot;ENABLE_DB_ACCESS Procedure&quot; on page 5-31</td>
<td>Grants an AQ Internet agent the privileges of a specific database user</td>
</tr>
<tr>
<td>&quot;DISABLE_DB_ACCESS Procedure&quot; on page 5-31</td>
<td>Revokes the privileges of a database user from an AQ Internet agent</td>
</tr>
<tr>
<td>&quot;ADD_ALIAS_TO_LDAP Procedure&quot; on page 5-32</td>
<td>Creates an alias for a queue, agent, or a JMSConnectionFactory in LDAP.</td>
</tr>
<tr>
<td>&quot;DEL_ALIAS_FROM_LDAP Procedure&quot; on page 5-32</td>
<td>Drops an alias for a queue, agent, or JMSConnectionFactory in LDAP.</td>
</tr>
</tbody>
</table>
- 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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Name of a queue table to be created.</td>
</tr>
<tr>
<td>queue_payload_type</td>
<td>Type of the user data stored. See &quot;Type Name&quot; on page 4-3 for valid values for this parameter.</td>
</tr>
</tbody>
</table>
CREATE_QUEUE_TABLE Procedure

Table 5–3  CREATE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage_clause</td>
<td>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, TABLESPACE, 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.</td>
</tr>
<tr>
<td>sort_list</td>
<td>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.</td>
</tr>
<tr>
<td>multiple_consumers</td>
<td>FALSE: Queues created in the table can only have one consumer per message. This is the default. TRUE: Queues created in the table can have multiple consumers per message.</td>
</tr>
</tbody>
</table>
CLOB, BLOB, and BFILE are valid attributes for AQ object type payloads. However, only CLOB and BLOB can be propagated using AQ propagation in Oracle8i release 8.1.5 or later. See the Oracle9i Application Developer’s Guide - Advanced Queuing for more information.

Usage Notes
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queue_table</code></td>
<td>Name of a queue table to be created.</td>
</tr>
<tr>
<td><code>comment</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>primary_instance</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>secondary_instance</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
**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 done automatically.

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Name of a queue table to be dropped.</td>
</tr>
</tbody>
</table>
| 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**

```sql
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,
```

---
CREATE_QUEUE Procedure

Parameters

Table 5–6  CREATE_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in the Oracle9i SQL Reference with regard to reserved characters.</td>
</tr>
<tr>
<td>queue_table</td>
<td>Name of the queue table that will contain the queue.</td>
</tr>
<tr>
<td>queue_type</td>
<td>Specifies whether the queue being created is an exception queue or a normal queue.</td>
</tr>
<tr>
<td></td>
<td>NORMAL_QUEUE: The queue is a normal queue. This is the default.</td>
</tr>
<tr>
<td></td>
<td>EXCEPTION_QUEUE: It is an exception queue. Only the dequeue operation is allowed on the exception queue.</td>
</tr>
<tr>
<td>max_retries</td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>The count is incremented when the application issues a rollback after executing the dequeue. The message is moved to the exception queue when it reaches its max_retries.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>retry_delay</td>
<td>Delay time, in seconds, before this message is scheduled for processing again after an application rollback.</td>
</tr>
<tr>
<td></td>
<td>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 retry_delay is supported for single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.</td>
</tr>
</tbody>
</table>
Summary of DBMS_AQADM Subprograms

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

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

Table 5–6  CREATE_QUEUE Procedure Parameters

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

DBMS_AQADM  5-11
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, then 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 ( 
```
Summary of DBMS_AQADM Subprograms

Parameters

Table 5–8 ALTER_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue that is to be altered.</td>
</tr>
<tr>
<td>max_retries</td>
<td>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.</td>
</tr>
<tr>
<td>retry_delay</td>
<td>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.</td>
</tr>
<tr>
<td>retention_time</td>
<td>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.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>TRUE: Causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_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.</td>
</tr>
</tbody>
</table>
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

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

Parameters

Table 5–9 DROP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue that is to be dropped.</td>
</tr>
</tbody>
</table>
| 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 and/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.
Summary of DBMS_AQADM Subprograms

Syntax

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

Parameters

Table 5–10 START_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue to be enabled.</td>
</tr>
<tr>
<td>enqueue</td>
<td>Specifies whether ENQUEUE should be enabled on this queue.</td>
</tr>
<tr>
<td>dequeue</td>
<td>Specifies whether DEQUEUE should be enabled on this queue.</td>
</tr>
</tbody>
</table>

TRUE: Enable ENQUEUE. This is the default.
FALSE: Do not alter the current setting.

STOP_QUEUE Procedure

This procedure disables enqueuing and/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

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

Parameters

Table 5–11 STOP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue to be disabled.</td>
</tr>
<tr>
<td>enqueue</td>
<td>Specifies whether ENQUEUE should be disabled on this queue. TRUE: Disable ENQUEUE. This is the default. FALSE: Do not alter the current setting.</td>
</tr>
<tr>
<td>dequeue</td>
<td>Specifies whether DEQUEUE should be disabled on this queue. TRUE: Disable DEQUEUE. This is the default. FALSE: Do not alter the current setting.</td>
</tr>
<tr>
<td>wait</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

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

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);  
```
GRANT_QUEUE_PRIVILEGE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>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.</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.</td>
</tr>
</tbody>
</table>

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The AQ queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.</td>
</tr>
<tr>
<td>grant_option</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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**

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

**Parameters**

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

**ADD_SUBSCRIBER Procedure**

This procedure adds a default subscriber to a queue.

**Syntax**

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

Table 5–16  ADD_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent on whose behalf the subscription is being defined.</td>
</tr>
<tr>
<td>rule</td>
<td>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.</td>
</tr>
<tr>
<td>transformation</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 as shown below;

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent on whose behalf the subscription is being altered. See &quot;Agent&quot; on page 4-4.</td>
</tr>
<tr>
<td>rule</td>
<td>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.</td>
</tr>
<tr>
<td>transformation</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure alters both the rule and the transformation for the subscriber. If you wish 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.
SCHEDULE_PROPAGATION Procedure

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent who is being removed. See &quot;Agent&quot; on page 4-4.</td>
</tr>
</tbody>
</table>

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Table 5–19 SCHEDULE_PROPAGATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
<tr>
<td>start_time</td>
<td>Initial start time for the propagation window for messages from the source queue to the destination.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.</td>
</tr>
<tr>
<td>next_time</td>
<td>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'.</td>
</tr>
<tr>
<td>latency</td>
<td>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.</td>
</tr>
</tbody>
</table>

**UN SCHEDULE_PROPAGATION Procedure**

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

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```sql
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 5-21 VERIFY_QUEUE_TYPES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>dest_queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
<tr>
<td>rc</td>
<td>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.</td>
</tr>
</tbody>
</table>

ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters parameters for a propagation schedule.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.</td>
</tr>
<tr>
<td>next_time</td>
<td>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 ( \text{SYSDATE} + 1 - \frac{\text{duration}}{86400} ).</td>
</tr>
<tr>
<td>latency</td>
<td>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.</td>
</tr>
</tbody>
</table>
**ENABLE_PROPAGATION_SCHEDULE Procedure**

This procedure enables a previously disabled propagation schedule.

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>

**DISABLE_PROPAGATION_SCHEDULE Procedure**

This procedure disables a propagation schedule.

**Syntax**

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

Parameters

Table 5–24 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

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

Parameters

Table 5–25 MIGRATE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Specifies name of the queue table to be migrated.</td>
</tr>
<tr>
<td>compatible</td>
<td>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.</td>
</tr>
</tbody>
</table>

CREATE_AQ_AGENT Procedure

This procedure registers an agent for AQ Internet access using HTTP/SMTP protocols.
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 5–26 CREATE_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the AQ Internet agent</td>
</tr>
<tr>
<td>certificate_location</td>
<td>Agent’s certificate location in LDAP (default= NULL). If the agent is allowed to access AQ via SMTP, then its certificate must be registered in LDAP. For access via HTTP, the certificate location is not required</td>
</tr>
<tr>
<td>enable_http</td>
<td>TRUE: the agent can access AQ via HTTP</td>
</tr>
<tr>
<td>enable_smtp</td>
<td>TRUE: the agent can access AQ via SMTP (email)</td>
</tr>
<tr>
<td>enable_anyp</td>
<td>TRUE: the agent can access AQ via any protocol (HTTP or SMTP)</td>
</tr>
</tbody>
</table>

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.

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,
DROP_AQ_AGENT Procedure

Parameters

Table 5–27 ALTER_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the AQ Internet agent</td>
</tr>
<tr>
<td>certification_location</td>
<td>Agent’s certificate location in LDAP (default= NULL).</td>
</tr>
<tr>
<td></td>
<td>If the agent is allowed to access AQ via SMTP, then its certificate must</td>
</tr>
<tr>
<td></td>
<td>be registered in LDAP.</td>
</tr>
<tr>
<td></td>
<td>For access via HTTP, the certificate location is not required</td>
</tr>
<tr>
<td>enable_http</td>
<td>TRUE: the agent can access AQ via HTTP</td>
</tr>
<tr>
<td></td>
<td>FALSE: the agent cannot access AQ via HTTP</td>
</tr>
<tr>
<td>enable_smtp</td>
<td>TRUE: the agent can access AQ via SMTP (email)</td>
</tr>
<tr>
<td></td>
<td>FALSE: the agent cannot access AQ via SMTP</td>
</tr>
<tr>
<td>enable_anyp</td>
<td>TRUE: the agent can access AQ via any protocol (HTTP or SMTP)</td>
</tr>
</tbody>
</table>

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 5–28 DROP_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the AQ Internet agent</td>
</tr>
</tbody>
</table>
**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.

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the AQ Internet agent</td>
</tr>
<tr>
<td>db_username</td>
<td>Specified the database user whose privileges are to be granted to the AQ Internet agent</td>
</tr>
</tbody>
</table>

**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)
```
ADD_ALIAS_TO_LDAP Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>the name of the alias</td>
</tr>
<tr>
<td>obj_location</td>
<td>The distinguished name of the object (queue, agent or connection factory) to which the above alias refers to</td>
</tr>
</tbody>
</table>

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.DEALIAS_FROM_LDAP(
    alias IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>The alias to be removed</td>
</tr>
</tbody>
</table>
DEL_ALIAS_FROM_LDAP Procedure
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
Summary of DBMS_AQELM Subprograms

Table 6–1  DBMS_AQELM Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET_MAILHOST Procedure&quot; on page 6-2</td>
<td>Sets the host name for SMTP server.</td>
</tr>
<tr>
<td>&quot;GET_MAILHOST Procedure&quot; on page 6-3</td>
<td>Gets the host name for SMTP server.</td>
</tr>
<tr>
<td>&quot;SET_MAILPORT Procedure&quot; on page 6-3</td>
<td>Sets the port number for SMTP server.</td>
</tr>
<tr>
<td>&quot;GET_MAILPORT Procedure&quot; on page 6-4</td>
<td>Gets the port number for SMTP server.</td>
</tr>
<tr>
<td>&quot;SET_SENDFROM Procedure&quot; on page 6-4</td>
<td>Sets the sent-from e-mail address.</td>
</tr>
<tr>
<td>&quot;GET_SENDFROM Procedure&quot; on page 6-5</td>
<td>Gets the sent-from e-mail address.</td>
</tr>
<tr>
<td>&quot;SET_PROXY Procedure&quot; on page 6-5</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;GET_PROXY Procedure&quot; on page 6-6</td>
<td>Gets the proxy server name and no_proxy_domains set by DBMS_AQELM.SET_PROXY for HTTP notifications.</td>
</tr>
</tbody>
</table>

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

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

Parameters

Table 6–2 shows the parameters for the SET_MAILHOST procedure.
Summary of DBMS_AQELM Subprograms

**GET_MAILHOST Procedure**

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

**Syntax**

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

**Parameters**

Table 6–3 shows the parameters for the GET_MAILHOST procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailhost</td>
<td>The SMTP server host name.</td>
</tr>
</tbody>
</table>

**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**

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

**Parameters**

Table 6–4 shows the parameters for the SET_MAILPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailport</td>
<td>The SMTP server port number.</td>
</tr>
</tbody>
</table>
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**

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

**Parameters**

Table 6–5 shows the parameters for the GET_MAILPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailport</td>
<td>The SMTP server port number.</td>
</tr>
</tbody>
</table>

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**

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

**Parameters**

Table 6–6 shows the parameters for the SET_SENDFROM procedure.
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 6–7 shows the parameters for the GET_SENDFROM procedure.

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendfrom</td>
<td>The sent-from e-mail address.</td>
</tr>
</tbody>
</table>
```

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 at. 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

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

Parameters

Table 6–8 shows the parameters for the SET_PROXY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>The proxy server host and port number. The syntax is &quot;[http://]host[:port][/]&quot;. For example, &quot;www-proxy.my-company.com:80&quot;.</td>
</tr>
<tr>
<td>no_proxy_domains</td>
<td>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, &quot;corp.my-company.com, eng.my-company.com:80&quot;</td>
</tr>
</tbody>
</table>

GET_PROXY Procedure

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

Syntax

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

Parameters

Table 6–9 shows the parameters for the GET_PROXY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>The proxy server host and port number.</td>
</tr>
</tbody>
</table>
### Table 6–9  GET_PROXY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_proxy_domains</td>
<td>The list of no-proxy domains or hosts.</td>
</tr>
</tbody>
</table>
GET_PROXY Procedure
The DBMS_BACKUP_RESTORE package has a PL/SQL procedure to normalize filenames on Windows NT platforms.

**Note:** Do not use this procedure on Oracle releases prior to 8.1.6 or on UNIX-based Oracle installations.

This chapter discusses the following topics:
- Filename Normalization for Oracle on Windows NT Platforms
Filename Normalization for Oracle on Windows NT Platforms

In release 8.1.6 and higher, Oracle correctly normalizes filenames. However, you must use this procedure to normalize filenames in the control file and recovery catalog from earlier releases.

In releases prior to 8.1.6, a flawed filename normalization mechanism allowed two different filenames to refer to one physical file. DBMS_BACKUP_RESTORE corrects this so that Oracle accurately identifies all physical files referenced in the control file and the recovery catalog.

For more information on this package and for detailed procedures on executing it, please refer to Oracle9i Database Migration.
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 calling user, rather than the package owner SYS.

This chapter discusses the following topics:

- Summary of DBMS_DDL Subprograms
ALTER_COMPILE Procedure

This procedure is equivalent to the following SQL statement:

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

If the named object is this package, or any packages upon which it depends (currently, STANDARD or DBMS_STANDARD), then the procedure simply returns, and these packages are successfully compiled.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER.</td>
</tr>
<tr>
<td>schema</td>
<td>Schema name.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then use current schema (case-sensitive).</td>
</tr>
<tr>
<td>name</td>
<td>Name of the object (case-sensitive).</td>
</tr>
</tbody>
</table>
Exceptions

Table 8–3  ALTER_COMPILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000:</td>
<td>Insufficient privileges or object does not exist.</td>
</tr>
<tr>
<td>ORA-20001:</td>
<td>Remote object, cannot compile.</td>
</tr>
<tr>
<td>ORA-20002:</td>
<td>Bad value for object type</td>
</tr>
<tr>
<td></td>
<td>Should be either PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, or TRIGGER.</td>
</tr>
</tbody>
</table>

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

```sql
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 8–4  ANALYZE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of TABLE, CLUSTER or INDEX. If none of these, an ORA-20001 error is raised.</td>
</tr>
<tr>
<td>schema</td>
<td>Schema of object to analyze. NULL means current schema, case-sensitive.</td>
</tr>
<tr>
<td>name</td>
<td>Name of object to analyze, case-sensitive.</td>
</tr>
</tbody>
</table>
ANALYZE_OBJECT Procedure

Table 8–4 ANALYZE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>One of ESTIMATE, COMPUTE or DELETE.</td>
</tr>
<tr>
<td></td>
<td>If ESTIMATE, then either estimate_rows or estimate_percent must be non-zero.</td>
</tr>
<tr>
<td>estimate_rows</td>
<td>Number of rows to estimate.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate.</td>
</tr>
<tr>
<td></td>
<td>If estimate_rows is specified, then ignore this parameter.</td>
</tr>
<tr>
<td>method_opt</td>
<td>Method options of the following format.</td>
</tr>
<tr>
<td></td>
<td>[ FOR TABLE ]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL [INDEXED] COLUMNS] [SIZE n]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL INDEXES ]</td>
</tr>
<tr>
<td>partname</td>
<td>Specific partition to be analyzed.</td>
</tr>
</tbody>
</table>

Exceptions

Table 8–5 ANALYZE_OBJECT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000:</td>
<td>Insufficient privileges or object does not exist.</td>
</tr>
<tr>
<td>ORA-20001:</td>
<td>Bad value for object type.</td>
</tr>
<tr>
<td></td>
<td>Should be either TABLE, INDEX or CLUSTER.</td>
</tr>
<tr>
<td>ORA-20002:</td>
<td>METHOD must be one of COMPUTE, ESTIMATE or DELETE.</td>
</tr>
</tbody>
</table>
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
- Usage Notes
- Types and Constants
- Error Codes, Exceptions, and Variables
- Common and Debug Session Sections
- OER Breakpoints
- Summary of DBMS_DEBUG Subprograms
Using DBMS_DEBUG

To debug server-side code, it is necessary to 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 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 what target session to supervise. The debug session may then call entrypoints in DBMS_DEBUG to read events that were posted from the target session and to communicate with the target session.

See Also: Figure 9–1 and Figure 9–2 illustrate the flow of operations in the session to be debugged and in the debugging session.

DBMS_DEBUG does not provide any 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 9–1 Target Session

- Initialize session for debugging, and generate/specify unique debugID. DBMS_DEBUG.initialize()
- Start debugging DBMS_DEBUG.debug_on()
- Execute PL/SQL programs
- Stop debugging DBMS_DEBUG.debug_off()
Manipulate breakpoints
DBMS_DEBUG.set_breakpoint()
DBMS_DEBUG.delete_breakpoint()
DBMS_DEBUG.disable_breakpoint()
DBMS_DEBUG.enable_breakpoint()
DBMS_DEBUG.show_breakpoints()

Read first event from target session
DBMS_DEBUG.synchronize()

Show stack
DBMS_DEBUG.print_backtrace()

Get/set values
DBMS_DEBUG.get_value()
DBMS_DEBUG.set_value()

Manipulate breakpoints

Show source
DBMS_DEBUG.show_source()
**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, then these parameters have no effect.

Types and Constants

Types

PROGRAM_INFO
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntrypointName</td>
<td>Null, unless this is a nested procedure or function.</td>
</tr>
<tr>
<td>LibunitType</td>
<td>Disambiguate among objects that share the same namespace (for example, procedure and package specifications).</td>
</tr>
</tbody>
</table>

See the Libunit Types on page 9-9 for more information.
Types and Constants

TYPE program_info IS RECORD
{
    -- The following fields are used when setting a breakpoint
    Namespace  BINARY_INTEGER, -- See 'NAMESPACE' 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
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
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
This type is used by GET_INDEXES to return the available indexes for an indexed table.

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

**BACKTRACE_TABLE**

This type is used by PRINT_BACKTRACE.

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

**BREAKPOINT_TABLE**

This type is used by SHOW_BREAKPOINTS.

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

**VC2_TABLE**

This type is used by SHOW_SOURCE.

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

**Constants**

A breakpoint status may have these values:

- `breakpoint_status_unused` Breakpoint is not in use.
- `breakpoint_status_active` A line breakpoint.
- `breakpoints_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.
Types and Constants

- 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.

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.

- 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.
### Information Flags

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

- `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_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_timeout` — A timeout occurred.
- `reasonInstantiate` — 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.

success Normal termination.

Statuses returned by GET_VALUE and SET_VALUE:

- 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:

- errorillegal_value: Constraint violation.
- errorillegal_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:

- error_no_such_breakp: No such breakpoint.
- error_idle_breakp: Cannot enable or disable an unused breakpoint.
General error codes (returned by many of the DBMS_DEBUG subprograms):

- **error_bad_handle**: Unable to set breakpoint in given program (non-existent or security violation).
- **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**: Timeout occurred.

**Exceptions**

- **illegal_init**: DEBUG_ON was called prior to INITIALIZE.

The following exceptions are raised by procedure SELF_CHECK:

- **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**

- **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**
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.
<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;PROBE_VERSION Procedure&quot; on page 9-15</td>
<td>Returns the version number of DBMS_DEBUG on the server.</td>
</tr>
<tr>
<td>&quot;SELF_CHECK Procedure&quot; on page 9-15</td>
<td>Performs an internal consistency check.</td>
</tr>
<tr>
<td>&quot;SET_TIMEOUT Function&quot; on page 9-16</td>
<td>Sets the timeout value.</td>
</tr>
<tr>
<td>&quot;INITIALIZE Function&quot; on page 9-17</td>
<td>Sets debugID in target session.</td>
</tr>
<tr>
<td>&quot;DEBUG_ON Procedure&quot; on page 9-18</td>
<td>Turns debug-mode on.</td>
</tr>
<tr>
<td>&quot;DEBUG_OFF Procedure&quot; on page 18</td>
<td>Turns debug-mode off.</td>
</tr>
<tr>
<td>&quot;ATTACH_SESSION Procedure&quot; on page 9-19</td>
<td>Notifies the debug session about the target debugID.</td>
</tr>
<tr>
<td>&quot;SYNCHRONIZE Function&quot; on page 9-19</td>
<td>Waits for program to start running.</td>
</tr>
<tr>
<td>&quot;SHOW_SOURCE Procedure&quot; on page 9-20</td>
<td>Fetches program source.</td>
</tr>
<tr>
<td>&quot;PRINT_BACKTRACE Procedure&quot; on page 9-22</td>
<td>Prints a stack backtrace.</td>
</tr>
<tr>
<td>&quot;CONTINUE Function&quot; on page 9-23</td>
<td>Continues execution of the target program.</td>
</tr>
<tr>
<td>&quot;SET_BREAKPOINT Function&quot; on page 9-24</td>
<td>Sets a breakpoint in a program unit.</td>
</tr>
<tr>
<td>&quot;DELETE_BREAKPOINT Function&quot; on page 9-25</td>
<td>Deletes a breakpoint.</td>
</tr>
<tr>
<td>&quot;DISABLE_BREAKPOINT Function&quot; on page 9-26</td>
<td>Disables a breakpoint.</td>
</tr>
<tr>
<td>&quot;ENABLE_BREAKPOINT Function&quot; on page 9-27</td>
<td>Activates an existing breakpoint.</td>
</tr>
<tr>
<td>&quot;SHOW_BREAKPOINTS Procedure&quot; on page 9-27</td>
<td>Returns a listing of the current breakpoints.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version number.</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version number: increments as functionality is added.</td>
</tr>
</tbody>
</table>

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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout to use for the communication test. Default is 60 seconds.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OER-6516</td>
<td>Probe version is inconsistent.</td>
</tr>
<tr>
<td>pipe_creation_failure</td>
<td>Could not create a pipe.</td>
</tr>
<tr>
<td>pipe_send_failure</td>
<td>Could not write data to the pipe.</td>
</tr>
<tr>
<td>pipe_receive_failure</td>
<td>Could not read data from the pipe.</td>
</tr>
<tr>
<td>pipe_datatype_mismatch</td>
<td>Datatype in the pipe was wrong.</td>
</tr>
<tr>
<td>pipe_data_error</td>
<td>Data got garbled in the pipe.</td>
</tr>
</tbody>
</table>

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**

```sql
DBMS_DEBUG.SET_TIMEOUT (    timeout BINARY_INTEGER)
```
**Parameters**

### Table 9–5  SET_TIMEOUT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout to use for communication between the target and debug sessions.</td>
</tr>
</tbody>
</table>

**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**

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

**Parameters**

### Table 9–6  INITIALIZE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_session_id</td>
<td>Name of session ID. If NULL, then a unique ID is generated.</td>
</tr>
</tbody>
</table>
| diagnostics    | Indicates whether to dump diagnostic output to the tracefile.  
                  0 = (default) no diagnostics  
                  1 = print diagnostics |
DEBUG ON Procedure

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 9–7  DEBUG ON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_client_side_plsql_engine</td>
<td>Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine.</td>
</tr>
<tr>
<td>immediate</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_session_id</td>
<td>Debug ID from a call to INITIALIZE in target session.</td>
</tr>
<tr>
<td>diagnostics</td>
<td>Generate diagnostic output if non-zero.</td>
</tr>
</tbody>
</table>

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_info</td>
<td>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.</td>
</tr>
</tbody>
</table>
The best way to get the source code (for a program that is being run) is to use SQL. For example:

```sql
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 non-persistent programs (for example, anonymous blocks and trigger invocation blocks). For non-persistent 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.

---

**Table 9–9 SYNCHRONIZE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_requested</td>
<td>Optional bit-field in which to request information other than the default (which is info_getStackSize + info_getLineInfo). 0 means that no information is requested at all.</td>
</tr>
</tbody>
</table>

See “Information Flags” on page 9-10.

**Table 9–10 SYNCHRONIZE Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>Timed out before the program started execution.</td>
</tr>
<tr>
<td>error_timeout</td>
<td>Other communication error.</td>
</tr>
</tbody>
</table>

**SHOW_SOURCE Procedure**

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

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

Parameters

Table 9–11  SHOW_SOURCE Procedure Parameters

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

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);
PRINT_BACKTRACE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_line</td>
<td>Smallest line-number to print.</td>
</tr>
<tr>
<td>last_line</td>
<td>Largest line-number to print.</td>
</tr>
<tr>
<td>window</td>
<td>'Window' of lines (the number of lines around the current source line).</td>
</tr>
<tr>
<td>print_arrow</td>
<td>Non-zero means to print an arrow before the current line.</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer in which to place the source listing.</td>
</tr>
<tr>
<td>buflen</td>
<td>Length of buffer.</td>
</tr>
<tr>
<td>pieces</td>
<td>Set to non-zero if not all the source could be placed into the given buffer.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing</td>
<td>A formatted character buffer with embedded newlines.</td>
</tr>
</tbody>
</table>

Syntax

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

**Table 9–14 PRINT_BACKTRACE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtrace</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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**

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

**Parameters**

**Table 9–15 CONTINUE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_info</td>
<td>Information about the state of the program.</td>
</tr>
<tr>
<td>breakflags</td>
<td>Mask of events that are of interest. See &quot;Breakflags&quot; on page 9-9.</td>
</tr>
<tr>
<td>info_requested</td>
<td>Which information should be returned in run_info when the program stops. See &quot;Information Flags&quot; on page 9-10.</td>
</tr>
</tbody>
</table>
SET_BREAKPOINT Function

Returns

| Table 9–16  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

```sql
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 9–17  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. |
Returns

Table 9–18  SET_BREAKPOINT Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
<tr>
<td>error_illegal_line</td>
<td>Cannot set a breakpoint at that line.</td>
</tr>
<tr>
<td>error_bad_handle</td>
<td>No such program unit exists.</td>
</tr>
</tbody>
</table>

DELETE_BREAKPOINT Function

This function deletes a breakpoint.

Syntax

```sql
DBMS_DEBUG.DELETE_BREAKPOINT(
    breakpoint IN BINARY_INTEGER)
RETURN BINARY_INTEGER;
```
DISABLE_BREAKPOINT Function

Parameters

Table 9–19  DELETE_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint</td>
<td>Breakpoint number from a previous call to SET_BREAKPOINT.</td>
</tr>
</tbody>
</table>

Returns

Table 9–20  DELETE_BREAKPOINT Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
<tr>
<td>error_no_such_breakpt</td>
<td>No such breakpoint exists.</td>
</tr>
<tr>
<td>error_idle_breakpt</td>
<td>Cannot delete an unused breakpoint.</td>
</tr>
<tr>
<td>error_stale_breakpt</td>
<td>The program unit was redefined since the breakpoint was set.</td>
</tr>
</tbody>
</table>

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 9–21  DISABLE_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint</td>
<td>Breakpoint number from a previous call to SET_BREAKPOINT.</td>
</tr>
</tbody>
</table>

Returns

Table 9–22  DISABLE_BREAKPOINT Function Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
</tbody>
</table>
Summary of DBMS_DEBUG Subprograms

---

**ENABLE_BREAKPOINT Function**

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

**Syntax**

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

**Parameters**

- **breakpoint**
  - Description: Breakpoint number from a previous call to `SET_BREAKPOINT`.

**Returns**

- **success**
- **error_no_such_breakpt**
  - Description: No such breakpoint exists.
- **error_idle_breakpt**
  - Description: 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**

```sql
DBMS_DEBUG.SHOW_BREAKPOINTS (
```

---
GET_VALUE Function

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

### Parameters

#### Table 9–25  SHOW_BREAKPOINTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing</td>
<td>A formatted buffer (including newlines) of the breakpoints.</td>
</tr>
</tbody>
</table>

### Syntax

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

### Parameters

#### Table 9–26  SHOW_BREAKPOINTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing</td>
<td>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.</td>
</tr>
</tbody>
</table>

### GET_VALUE Function

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

#### Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Name of the variable or parameter.</td>
</tr>
<tr>
<td>frame#</td>
<td>Frame in which it lives; 0 means the current procedure.</td>
</tr>
<tr>
<td>scalar_value</td>
<td>Value.</td>
</tr>
<tr>
<td>format</td>
<td>Optional date format to use, if meaningful.</td>
</tr>
</tbody>
</table>

Syntax

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

Parameters

Table 9–29  GET_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Name of the variable or parameter.</td>
</tr>
<tr>
<td>handle</td>
<td>Description of the package containing the variable.</td>
</tr>
<tr>
<td>scalar_value</td>
<td>Value.</td>
</tr>
<tr>
<td>format</td>
<td>Optional date format to use, if meaningful.</td>
</tr>
</tbody>
</table>

Returns

Table 9–30  GET_VALUE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_no_such_object</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- Package does not exist.</td>
</tr>
<tr>
<td></td>
<td>- Package is not instantiated.</td>
</tr>
<tr>
<td></td>
<td>- User does not have privileges to debug the package.</td>
</tr>
<tr>
<td></td>
<td>- Object does not exist in the package.</td>
</tr>
<tr>
<td>error_indexed_table</td>
<td>The object is a table, but no index was provided.</td>
</tr>
</tbody>
</table>

Example

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```sql
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;
```

9-30  Supplied PL/SQL Packages and Types Reference
**SET_VALUE Function**

This function sets a value in the currently-running program. There are two overloaded SET_VALUE functions.

**Syntax**

```sql
DBMS_DEBUG.SET_VALUE (  
    frame# IN binary_integer,  
    assignment_statement IN varchar2) 
RETURN BINARY_INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>frame#</code></td>
<td>Frame in which the value is to be set; 0 means the currently executing frame.</td>
</tr>
<tr>
<td><code>assignment_statement</code></td>
<td>An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, <code>x := 3;</code>. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>success</code></td>
<td></td>
</tr>
<tr>
<td><code>error_illegal_value</code></td>
<td>Not possible to set it to that value.</td>
</tr>
<tr>
<td><code>error_illegal_null</code></td>
<td>Cannot set to <code>NULL</code> because object type specifies it as ‘not null’.</td>
</tr>
<tr>
<td><code>error_value_malformed</code></td>
<td>Value is not a scalar.</td>
</tr>
<tr>
<td><code>error_name_incomplete</code></td>
<td>The assignment statement does not resolve to a scalar. For example, ‘x := 3;’, if x is a record.</td>
</tr>
</tbody>
</table>

This form of `SET_VALUE` sets the value of a package variable.
SET_VALUE Function

Syntax

```sql
DBMS_DEBUG.SET_VALUE (  
    handle IN program_info,  
    assignment_statement IN VARCHAR2)  
RETURN BINARY_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Description of the package containing the variable.</td>
</tr>
<tr>
<td>assignment_statement</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 9–34  SET_VALUE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_no_such_object</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- Package does not exist.</td>
</tr>
<tr>
<td></td>
<td>- Package is not instantiated.</td>
</tr>
<tr>
<td></td>
<td>- User does not have privileges to debug the package.</td>
</tr>
<tr>
<td></td>
<td>- Object does not exist in the package.</td>
</tr>
</tbody>
</table>

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:

```sql
DECLARE  
    handle  dbms_debug.program_info;  
    retval  BINARY_INTEGER;  
BEGIN  
    handle.Owner := 'SCOTT';
```
Summary of DBMS_DEBUG Subprograms

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 9–35 GET_RUNTIME_INFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_requested</td>
<td>Which information should be returned in run_info when the program stops. See &quot;Information Flags&quot; on page 9-10.</td>
</tr>
<tr>
<td>run_info</td>
<td>Information about the state of the program.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_DEBUG.GET_INDEXES (
    varname  IN VARCHAR2,
    frame#   IN BINARY_INTEGER,
    handle   IN program_info,
    entries  OUT index_table)
RETURN BINARY_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varname</td>
<td>Name of the variable to get index information about.</td>
</tr>
<tr>
<td>frame#</td>
<td>Number of frame in which the variable or parameter resides; NULL for a package variable.</td>
</tr>
<tr>
<td>handle</td>
<td>Package description, if object is a package variable.</td>
</tr>
<tr>
<td>entries</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_no_such_object</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- The package does not exist.</td>
</tr>
<tr>
<td></td>
<td>- The package is not instantiated.</td>
</tr>
<tr>
<td></td>
<td>- The user does not have privileges to debug the package.</td>
</tr>
<tr>
<td></td>
<td>- The object does not exist in the package.</td>
</tr>
</tbody>
</table>
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

```sql
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 9–38 EXECUTE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>SQL or PL/SQL source to execute.</td>
</tr>
<tr>
<td>frame#</td>
<td>The context in which to execute the code. Only -1 (global context) is supported at this time.</td>
</tr>
<tr>
<td>bind_results</td>
<td>Whether the source wants to bind to results in order to return values from the target session.</td>
</tr>
<tr>
<td>0 = No</td>
<td></td>
</tr>
<tr>
<td>1 = Yes</td>
<td></td>
</tr>
<tr>
<td>results</td>
<td>Collection in which to place results, if bind_results is not 0.</td>
</tr>
<tr>
<td>errm</td>
<td>Error message, if an error occurred; otherwise, NULL.</td>
</tr>
</tbody>
</table>

Example 1

This example executes a SQL statement. It returns no results.

```sql
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);
```

---

**Summary of DBMS_DEBUG Subprograms**
EXECUTE Procedure

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.EXEND(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; ' ||
  )
```

9-36  Supplied PL/SQL Packages and Types Reference
' :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.

Parameters

- pkgs the instantiated packages (OUT)
- 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
PROCEDURE print_instantiations (pkgs IN OUT NOCOPY backtrace_table, flags IN BINARY_INTEGER);

TARGET_PROGRAM_RUNNING Procedure

Return TRUE if the target session is currently executing a stored procedure, or FALSE if it is not.

FUNCTION target_program_running RETURN BOOLEAN;

PING Procedure

Ping 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.

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.

PROCEDURE ping;

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_BEHAVIOUR Procedure
This procedure tells Probe what to do with the target session when a timeout occurs. [This call is made in the target session.]

Parameters
- behavior - one of the following (see descriptions above):
  - retry_on_timeout
  - continue_on_timeout
  - nodebug_on_timeout
  - abort_on_timeout

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 re-establish control (at the next event) but does not cause the target session to hang indefinitely.

PROCEDURE set_timeout_behavior (behavior IN PLS_INTEGER);

GET_TIMEOUT_BEHAVIOUR - Returns the current timeout behavior. [This call is made in the target session.]

FUNCTION 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
  );

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

  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:
Summary of DBMS_DEBUG Subprograms

- 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 via delete_oer_breakpoint.

FUNCTION set_oer_breakpoint(oer IN PLS_INTEGER) RETURN PLS_INTEGER;
   DELETE_OER_BREAKPOINT
       Delete an OER breakpoint.

Parameters

   oer - the OER (positive 4-byte number) to delete

Returns

   success
   error_no_such_breakpt - no such OER breakpoint exists

FUNCTION delete_oer_breakpoint(oer IN PLS_INTEGER) RETURN PLS_INTEGER;
   SHOW_BREAKPOINTS

Parameters

   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);
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 10–1 DBMS_DEFER Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CALL Procedure&quot; on page 10-3</td>
<td>Builds a deferred call to a remote procedure.</td>
</tr>
<tr>
<td>&quot;COMMIT_WORK Procedure&quot; on page 10-4</td>
<td>Performs a transaction commit after checking for well-formed deferred remote procedure calls.</td>
</tr>
<tr>
<td>&quot;datatype_ARG Procedure&quot; on page 10-5</td>
<td>Provides the data that is to be passed to a deferred remote procedure call.</td>
</tr>
<tr>
<td>&quot;TRANSACTION Procedure&quot; on page 10-8</td>
<td>Indicates the start of a new deferred transaction.</td>
</tr>
</tbody>
</table>
CALL Procedure

This procedure builds a deferred call to a remote procedure.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema in which the stored procedure is located.</td>
</tr>
<tr>
<td>package_name</td>
<td>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.</td>
</tr>
<tr>
<td>proc_name</td>
<td>Name of the remote procedure to which you want to defer a call.</td>
</tr>
<tr>
<td>arg_count</td>
<td>Number of parameters for the procedure. You must have one call to <code>DBMS_DEFER.datatype_ARG</code> for each of these parameters. <strong>Note:</strong> You must include all of the parameters for the procedure, even if some of the parameters have defaults.</td>
</tr>
<tr>
<td>nodes</td>
<td>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 <code>no_data_found</code> exception is raised. The data in the table is case insensitive. This parameter is optional.</td>
</tr>
<tr>
<td>group_name</td>
<td>Reserved for internal use.</td>
</tr>
</tbody>
</table>
COMMIT_WORK Procedure

Exceptions

Table 10–3  CALL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23304</td>
<td>Previous call was not correctly formed.</td>
</tr>
<tr>
<td>(malformedcall)</td>
<td></td>
</tr>
<tr>
<td>ORA-23319</td>
<td>Parameter value is not appropriate.</td>
</tr>
<tr>
<td>ORA-23352</td>
<td>Destination list (specified by nodes or by a previous DBMS_DEFER.TRANSACTION call) contains duplicates.</td>
</tr>
</tbody>
</table>

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 10–4  COMMIT_WORK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit_work_comment</td>
<td>Equivalent to the COMMIT COMMENT statement in SQL.</td>
</tr>
</tbody>
</table>

Exceptions

Table 10–5  COMMIT_WORK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23304</td>
<td>Transaction was not correctly formed or terminated.</td>
</tr>
<tr>
<td>(malformedcall)</td>
<td></td>
</tr>
</tbody>
</table>
**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:

```sql
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:

```sql
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 *REF*s. See *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* 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 1-7.

**Syntax**

```sql
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 10–6  datatype_ARG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>Value of the parameter that you want to pass to the remote procedure to which you previously deferred a call.</td>
</tr>
</tbody>
</table>

Exceptions

Table 10–7  datatype_ARG Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23323</td>
<td>Argument value is too long.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23304 (malformedcall)</td>
<td>Previous transaction was not correctly formed or terminated.</td>
</tr>
<tr>
<td>ORA-23319</td>
<td>Parameter value is not appropriate.</td>
</tr>
<tr>
<td>ORA-23352</td>
<td>Raised by DBMS_DEFER.CALL if the node list contains duplicates.</td>
</tr>
</tbody>
</table>
DBMS_DEFER_QUERY enables querying 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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GET_ARG_FORM Function&quot; on page 11-3</td>
<td>Determines the form of an argument in a deferred call.</td>
</tr>
<tr>
<td>&quot;GET_ARG_TYPE Function&quot; on page 11-5</td>
<td>Determines the type of an argument in a deferred call.</td>
</tr>
<tr>
<td>&quot;GET_CALL_ARGS Procedure&quot; on page 11-7</td>
<td>Returns the text version of the various arguments for the specified call.</td>
</tr>
<tr>
<td>&quot;GET_datatype_ARG Function&quot; on page 11-9</td>
<td>Determines the value of an argument in a deferred call.</td>
</tr>
<tr>
<td>&quot;GET_OBJECT_NULL_VECTOR_ARG Function&quot; on page 11-12</td>
<td>Returns the type information for a column object.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_DEFER_QUERY.GET_ARG_FORM (  
callno IN NUMBER,  
arg_no IN NUMBER,  
deferred_tran_id IN VARCHAR2)  
RETURN NUMBER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callno</td>
<td>Call identifier from the DEFCALL view.</td>
</tr>
<tr>
<td>arg_no</td>
<td>Position of desired parameter in calls argument list. Parameter positions are 1..number of parameters in call.</td>
</tr>
<tr>
<td>deferred_tran_id</td>
<td>Deferred transaction identification.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Input parameters do not correspond to a parameter of a deferred call.</td>
</tr>
</tbody>
</table>
**Returns**

<table>
<thead>
<tr>
<th>Constant Return Value</th>
<th>Return Value</th>
<th>Possible Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_DEFER_QUERY.ARG_FORM_NONE</td>
<td>0</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROWID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RAW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BLOB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User-defined types</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_FORM_IMPLICIT</td>
<td>1</td>
<td>CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARCHAR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLOB</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_FORM_NCHAR</td>
<td>2</td>
<td>NCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NVARCHAR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NCLOB</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_DEFER_QUERY.GET_ARG_TYPE (  
callno IN NUMBER,  
arg_no IN NUMBER,  
deferred_tran_id IN VARCHAR2)  
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callno</td>
<td>Identification number from the DEFCALL view of the deferred remote procedure call.</td>
</tr>
<tr>
<td>arg_no</td>
<td>Numerical position of the argument to the call whose type you want to determine. The first argument to a procedure is in position 1.</td>
</tr>
<tr>
<td>deferred_tran_id</td>
<td>Identifier of the deferred transaction.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Input parameters do not correspond to a parameter of a deferred call.</td>
</tr>
</tbody>
</table>
## GET_ARG_TYPE Function

### Returns

Table 11–7  GET_ARG_TYPE Function Returns

<table>
<thead>
<tr>
<th>Constant Return Value</th>
<th>Return Value</th>
<th>Corresponding Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_VARCHAR2</td>
<td>1</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_NUM</td>
<td>2</td>
<td>NUMBER</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_ROWID</td>
<td>11</td>
<td>ROWID</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_DATE</td>
<td>12</td>
<td>DATE</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_RAW</td>
<td>23</td>
<td>RAW</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_CHAR</td>
<td>96</td>
<td>CHAR</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_AnyData</td>
<td>109</td>
<td>AnyData</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_CLOB</td>
<td>112</td>
<td>CLOB</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_BLOB</td>
<td>113</td>
<td>BLOB</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_BFIL</td>
<td>114</td>
<td>BFILE</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_OBJECT_NULL_VECTOR</td>
<td>121</td>
<td>OBJECT_NULL_VECTOR</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_TIMESTAMP</td>
<td>180</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_TSTZ</td>
<td>181</td>
<td>TSTZ</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_IYM</td>
<td>182</td>
<td>IYM</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_IDS</td>
<td>183</td>
<td>IDS</td>
</tr>
<tr>
<td>DBMS_DEFER_QUERY.ARG_TYPE_TSLTZ</td>
<td>231</td>
<td>TSLTZ</td>
</tr>
</tbody>
</table>
Summary of DBMS_DEFER_QUERY Subprograms

---

**Note:**

- The *AnyData* datatype supports the following user-defined types: object types, collections, and *REF*s. See *Oracle9i SQL Reference* and *Oracle9i Application Developer’s Guide - Object-Relational Features* 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 1-7.

---

**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 11-9
- *Oracle9i SQL Reference* and *Oracle9i Application Developer’s Guide - Object-Relational Features* for more information about the *AnyData* datatype

**Syntax**

```sql
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);
```
GET_CALL_ARGS Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callno</td>
<td>Identification number from the DEFCALL view of the deferred remote procedure call (RPC).</td>
</tr>
<tr>
<td>startarg</td>
<td>Numerical position of the first argument you want described.</td>
</tr>
<tr>
<td>argcnt</td>
<td>Number of arguments in the call.</td>
</tr>
<tr>
<td>argsize</td>
<td>Maximum size of returned argument.</td>
</tr>
<tr>
<td>tran_id</td>
<td>Identifier of the deferred transaction.</td>
</tr>
<tr>
<td>date_fmt</td>
<td>Format in which the date is returned.</td>
</tr>
<tr>
<td>types</td>
<td>Array containing the types of arguments.</td>
</tr>
<tr>
<td>forms</td>
<td>Array containing the character set forms of arguments.</td>
</tr>
<tr>
<td>vals</td>
<td>Array containing the values of the arguments in a textual form.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Input parameters do not correspond to a parameter of a deferred call.</td>
</tr>
</tbody>
</table>
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 10-5
- The Replication Management tool’s online help for information about displaying deferred transactions and error transactions in the Replication Management tool
- Oracle9i SQL Reference and Oracle9i Application Developer’s Guide - Object-Relational Features 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 1-7.
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 }```
Summary of DBMS_DEFER_QUERY Subprograms

Parameters

Table 11–10  GET_datatype_ARG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callno</td>
<td>Identification number from the DEFCALL view of the deferred remote procedure call.</td>
</tr>
<tr>
<td>arg_no</td>
<td>Numerical position of the argument to the call whose value you want to determine. The first argument to a procedure is in position 1.</td>
</tr>
<tr>
<td>deferred_tran_id</td>
<td>Identifier of the deferred transaction. Defaults to the last transaction identifier passed to the GET_ARG_TYPE function. The default is NULL.</td>
</tr>
</tbody>
</table>

Exceptions

Table 11–11  GET_datatype_ARG Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Input parameters do not correspond to a parameter of a deferred call.</td>
</tr>
<tr>
<td>ORA-26564</td>
<td>Argument in this position is not of the specified type or is not one of the types supported by the AnyData type.</td>
</tr>
</tbody>
</table>
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.REPCATS$OBJECT_NULL_VECTOR;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callno</td>
<td>Call identifier from the DEFCALL view.</td>
</tr>
<tr>
<td>arg_no</td>
<td>Position of desired parameter in calls argument list. Parameter positions are 1..number of parameters in call.</td>
</tr>
<tr>
<td>deferred_tran_id</td>
<td>Deferred transaction identification.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Input parameters do not correspond to a parameter of a deferred call.</td>
</tr>
<tr>
<td>ORA-26564</td>
<td>Parameter is not an object_null_vector type.</td>
</tr>
</tbody>
</table>
Returns

Table 11–14  GET_OBJECT_NULL_VECTOR_ARG Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Type Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.REPCAT$_OBJECT_NULL_VECTOR type</td>
<td>CREATE TYPE SYSTEM.REPCAT$_OBJECT_NULL_VECTOR</td>
</tr>
<tr>
<td></td>
<td>AS OBJECT (</td>
</tr>
<tr>
<td></td>
<td>type_owner VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>type_name VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>type_hashcode RAW(17),</td>
</tr>
<tr>
<td></td>
<td>null_vector RAW(2000));</td>
</tr>
</tbody>
</table>
GET_OBJECT_NULL_VECTOR_ARG Function
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
### Summary of DBMS_DEFER_SYS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_DEFAULT_DEST Procedure&quot; on page 12-4</td>
<td>Adds a destination database to the DEFDEFAULTDEST view.</td>
</tr>
<tr>
<td>&quot;CLEAR_PROP_STATISTICS Procedure&quot; on page 12-5</td>
<td>Clears the propagation statistics in the DEFSCHEDULE data dictionary view.</td>
</tr>
<tr>
<td>&quot;DELETE_DEFAULT_DEST Procedure&quot; on page 12-6</td>
<td>Removes a destination database from the DEFDEFAULTDEST view.</td>
</tr>
<tr>
<td>&quot;DELETE_DEFDESTINATION Procedure&quot; on page 12-6</td>
<td>Removes a destination database from the DEFSCHEDULE view.</td>
</tr>
<tr>
<td>&quot;DELETE_ERROR Procedure&quot; on page 12-7</td>
<td>Deletes a transaction from the DEFERROR view.</td>
</tr>
<tr>
<td>&quot;DELETE_TRAN Procedure&quot; on page 12-8</td>
<td>Deletes a transaction from the DEFTRANDEST view.</td>
</tr>
<tr>
<td>&quot;DISABLED Function&quot; on page 12-9</td>
<td>Determines whether propagation of the deferred transaction queue from the current site to a specified site is enabled.</td>
</tr>
<tr>
<td>&quot;EXCLUDE_PUSH Function&quot; on page 12-10</td>
<td>Acquires an exclusive lock that prevents deferred transaction PUSH.</td>
</tr>
<tr>
<td>&quot;EXECUTE_ERROR Procedure&quot; on page 12-11</td>
<td>Reexecutes a deferred transaction that did not initially complete successfully in the security context of the original receiver of the transaction.</td>
</tr>
<tr>
<td>&quot;EXECUTE_ERROR_AS_USER Procedure&quot; on page 12-12</td>
<td>Reexecutes a deferred transaction that did not initially complete successfully in the security context of the user who executes this procedure.</td>
</tr>
<tr>
<td>&quot;PURGE Function&quot; on page 12-13</td>
<td>Purges pushed transactions from the deferred transaction queue at your current master site or materialized view site.</td>
</tr>
<tr>
<td>&quot;PUSH Function&quot; on page 12-16</td>
<td>Forces a deferred remote procedure call queue at your current master site or materialized view site to be pushed to a remote site.</td>
</tr>
</tbody>
</table>
Summary of DBMS_DEFER_SYS Subprograms

Table 12–1  DBMS_DEFER_SYS Package Subprograms  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;REGISTER_PROPAGATOR Procedure&quot; on page 12-19</td>
<td>Registers the specified user as the propagator for the local database.</td>
</tr>
<tr>
<td>&quot;SCHEDULE_PURGE Procedure&quot; on page 12-20</td>
<td>Schedules a job to purge pushed transactions from the deferred transaction queue at your current master site or materialized view site.</td>
</tr>
<tr>
<td>&quot;SCHEDULE_PUSH Procedure&quot; on page 12-22</td>
<td>Schedules a job to push the deferred transaction queue to a remote site.</td>
</tr>
<tr>
<td>&quot;SET_DISABLED Procedure&quot; on page 12-24</td>
<td>Disables or enables propagation of the deferred transaction queue from the current site to a specified destination site.</td>
</tr>
<tr>
<td>&quot;UNREGISTER_PROPAGATOR Procedure&quot; on page 12-26</td>
<td>Unregisters a user as the propagator from the local database.</td>
</tr>
<tr>
<td>&quot;UNSCHEDULE_PURGE Procedure&quot; on page 12-27</td>
<td>Stops automatic purges of pushed transactions from the deferred transaction queue at a master site or materialized view site.</td>
</tr>
<tr>
<td>&quot;UNSCHEDULE_PUSH Procedure&quot; on page 12-27</td>
<td>Stops automatic pushes of the deferred transaction queue from a master site or materialized view site to a remote site.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>The fully qualified database name of the node that you want to add to the DEFDEFAULTDEST view.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23352</td>
<td>The dblink that you specified is already in the default list.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>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.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
<tr>
<td>force</td>
<td>When set to true, Oracle ignores all safety checks and deletes the destination.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred_tran_id</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred_tran_id</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_DEFER_SYS.DISABLED (
    destination IN VARCHAR2)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>The fully qualified database name of the node whose propagation status you want to check.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Propagation to this site from the current site is disabled.</td>
</tr>
<tr>
<td>false</td>
<td>Propagation to this site from the current site is enabled.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>Specified destination does not appear in the DEFSCHEDULE view.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_DEFER_SYS.EXCLUDE_PUSH (
    timeout IN INTEGER
) RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success, lock acquired.</td>
</tr>
<tr>
<td>1</td>
<td>Timeout, no lock acquired.</td>
</tr>
<tr>
<td>2</td>
<td>Deadlock, no lock acquired.</td>
</tr>
<tr>
<td>4</td>
<td>Already own lock.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_DEFER_SYS.EXECUTE_ERROR (    deferred_tran_id IN VARCHAR2,    destination IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred_tran_id</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24275 error</td>
<td>Illegal combinations of NULL and non-NULL parameters were used.</td>
</tr>
<tr>
<td>badparam</td>
<td>Parameter value missing or invalid (for example, if destination is NULL).</td>
</tr>
<tr>
<td>missinguser</td>
<td>Invalid user.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
    deferred_tran_id IN VARCHAR2,
    destination IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred_tran_id</td>
<td>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.</td>
</tr>
<tr>
<td>destination</td>
<td>The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24275error</td>
<td>Illegal combinations of NULL and non-NULL parameters were used.</td>
</tr>
<tr>
<td>badparam</td>
<td>Parameter value missing or invalid (for example, if destination is NULL).</td>
</tr>
<tr>
<td>missinguser</td>
<td>Invalid user.</td>
</tr>
</tbody>
</table>
Summary of DBMS_DEFER_SYS Subprograms

PURGE Function

This function purges pushed transactions from the deferred transaction queue at your current master site or materialized view site.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge_method</td>
<td>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 use 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 &quot;Usage Notes&quot; on page 12-15 for more information.</td>
</tr>
<tr>
<td>rollback_segment</td>
<td>Name of rollback segment to use for the purge, or NULL for default.</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.</td>
</tr>
</tbody>
</table>
Table 12–18  PURGE Function Parameters  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_seconds</td>
<td>If &gt; 0, then stop purge cleanly after the specified number of seconds of real time.</td>
</tr>
<tr>
<td>delay_seconds</td>
<td>Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.</td>
</tr>
<tr>
<td>transaction_count</td>
<td>If &gt; 0, then shut down cleanly after purging transaction_count number of transactions.</td>
</tr>
<tr>
<td>write_trace</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 12–19  Purge Function Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_ok</td>
<td>OK, terminated after delay_seconds expired.</td>
</tr>
<tr>
<td>result_startup_seconds</td>
<td>Terminated by lock timeout while starting.</td>
</tr>
<tr>
<td>result_execution_seconds</td>
<td>Terminated by exceeding execution_seconds.</td>
</tr>
<tr>
<td>result_transaction_count</td>
<td>Terminated by exceeding transaction_count.</td>
</tr>
<tr>
<td>result_errors</td>
<td>Terminated after errors.</td>
</tr>
<tr>
<td>result_split_del_order_limit</td>
<td>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.</td>
</tr>
<tr>
<td>result_purge_disabled</td>
<td>Queue purging is disabled internally for synchronization when adding new master sites without quiesce.</td>
</tr>
</tbody>
</table>
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argoutofrange</td>
<td>Parameter value is out of a valid range.</td>
</tr>
<tr>
<td>executiondisabled</td>
<td>Execution of purging is disabled.</td>
</tr>
<tr>
<td>defererror</td>
<td>Internal error.</td>
</tr>
</tbody>
</table>

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 `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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>The fully qualified database name of the master site or master materialized view site to which you are forwarding changes.</td>
</tr>
<tr>
<td>parallelism</td>
<td>0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>( n &gt; 1 ) specifies parallel propagation with ( n ) parallel processes.</td>
</tr>
<tr>
<td></td>
<td>1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>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.</td>
</tr>
<tr>
<td>stop_on_error</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Table 12–21  PUSH Function Parameters**  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_trace</td>
<td>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.</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>Maximum number of seconds to wait for a previous push to the same destination.</td>
</tr>
<tr>
<td>execution_seconds</td>
<td>If &gt; 0, then stop push cleanly after the specified number of seconds of real time. If <code>transaction_count</code> and <code>execution_seconds</code> are zero (the default), then transactions are executed until there are no more in the queue. The <code>execution_seconds</code> 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 <code>execution_seconds</code> 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 <code>transaction_count</code> or <code>delivery_order_limit</code> parameters.</td>
</tr>
<tr>
<td>delay_seconds</td>
<td>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.</td>
</tr>
<tr>
<td>transaction_count</td>
<td>If &gt; 0, then the maximum number of transactions to be pushed before stopping. If <code>transaction_count</code> and <code>execution_seconds</code> are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.</td>
</tr>
<tr>
<td>delivery_order_limit</td>
<td>Stop execution cleanly before pushing a transaction where <code>delivery_order &gt;= delivery_order_limit</code></td>
</tr>
</tbody>
</table>
## PUSH Function

### Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_ok</td>
<td>OK, terminated after delay_seconds expired.</td>
</tr>
<tr>
<td>result_startup_seconds</td>
<td>Terminated by lock timeout while starting.</td>
</tr>
<tr>
<td>result_execution_seconds</td>
<td>Terminated by exceeding execution_seconds.</td>
</tr>
<tr>
<td>result_transaction_count</td>
<td>Terminated by exceeding transaction_count.</td>
</tr>
<tr>
<td>result_delivery_order_limit</td>
<td>Terminated by exceeding delivery_order_limit.</td>
</tr>
<tr>
<td>result_errors</td>
<td>Terminated after errors.</td>
</tr>
<tr>
<td>result_push_disabled</td>
<td>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.</td>
</tr>
<tr>
<td>result_split_del_order_limit</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incompleteparallelpush</td>
<td>Serial propagation requires that parallel propagation shuts down cleanly.</td>
</tr>
<tr>
<td>executiondisabled</td>
<td>Execution of deferred remote procedure calls (RPCs) is disabled at the destination.</td>
</tr>
<tr>
<td>crt_err_err</td>
<td>Error while creating entry in DEFERROR.</td>
</tr>
<tr>
<td>deferred_rpc_quiesce</td>
<td>Replication activity for replication group is suspended.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Communication failure during deferred remote procedure call (RPC).</td>
</tr>
<tr>
<td>missingpropagator</td>
<td>A propagator does not exist.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
    username IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missinguser</td>
<td>Specified user does not exist.</td>
</tr>
<tr>
<td>alreadypropagator</td>
<td>Specified user is already the propagator.</td>
</tr>
<tr>
<td>duplicatepropagator</td>
<td>There is already a different propagator.</td>
</tr>
</tbody>
</table>
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

```sql
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 12–26  SCHEDULE_PURGE Procedure Parameters  (Page 1 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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`.
Summary of DBMS_DEFER_SYS Subprograms

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.

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:

\[ \text{dbms_defer_sys.purge_method_quick} \]

Specify the following for this parameter to use purge_method_precise:

\[ \text{dbms_defer_sys.purge_method_precise} \]

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 12-15. 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_date</td>
<td>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.</td>
</tr>
<tr>
<td>reset</td>
<td>Set to true to reset LAST_TXN_COUNT, LAST_ERROR, and LAST_MSG to NULL.</td>
</tr>
<tr>
<td>purge_method</td>
<td>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: [ \text{dbms_defer_sys.purge_method_quick} ] Specify the following for this parameter to use purge_method_precise: [ \text{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 &quot;Usage Notes&quot; on page 12-15. These usage notes are for the DBMS_DEFER_SYS.PURGE function, but they also apply to the DBMS_DEFER_SYS.SCHEDULE_PURGE procedure.</td>
</tr>
<tr>
<td>rollback_segment</td>
<td>Name of rollback segment to use for the purge, or NULL for default.</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.</td>
</tr>
<tr>
<td>execution_seconds</td>
<td>If &gt;0, then stop purge cleanly after the specified number of seconds of real time.</td>
</tr>
<tr>
<td>delay_seconds</td>
<td>Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.</td>
</tr>
<tr>
<td>transaction_count</td>
<td>If &gt;0, then shut down cleanly after purging transaction_count number of transactions.</td>
</tr>
<tr>
<td>write_trace</td>
<td>When set to true, Oracle records the result value returned by the PURGE function in the server's trace file.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>The fully qualified database name of the master site or master materialized view site to which you are forwarding changes.</td>
</tr>
<tr>
<td>interval</td>
<td>Allows you to provide a function to calculate the next time to push. This value is stored in the interval field of the DEF_SCHEDULE 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.</td>
</tr>
</tbody>
</table>

Table 12–27 SCHEDULE_PUSH Procedure Parameters (Page 1 of 2)
Table 12–27  SCHEDULE_PUSH Procedure Parameters  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_date</td>
<td>Allows you to specify a time to push deferred transactions to the remote site. This value is stored in the <code>next_date</code> field of the <code>DEFSCHEDULE</code> 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 <code>interval</code>.</td>
</tr>
<tr>
<td>reset</td>
<td>Set to <code>true</code> to reset <code>LAST_TXN_COUNT</code>, <code>LST_ERROR</code>, and <code>LAST_MSG</code> to <code>NULL</code>.</td>
</tr>
<tr>
<td>parallelism</td>
<td>0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>( n &gt; 1 ) specifies parallel propagation with ( n ) parallel processes.</td>
</tr>
<tr>
<td></td>
<td>1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>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.</td>
</tr>
<tr>
<td>stop_on_error</td>
<td>The default, <code>false</code>, indicates that the executor should continue even if errors, such as conflicts, are encountered. If <code>true</code>, then stops propagation at the first indication that a transaction encountered an error at the destination site.</td>
</tr>
<tr>
<td>write_trace</td>
<td>When set to <code>true</code>, Oracle records the result value returned by the function in the server’s trace file.</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>Maximum number of seconds to wait for a previous push to the same destination.</td>
</tr>
<tr>
<td>execution_seconds</td>
<td>If ( &gt;0 ), then stop execution cleanly after the specified number of seconds of real time. If <code>transaction_count</code> and <code>execution_seconds</code> are zero (the default), then transactions are executed until there are no more in the queue.</td>
</tr>
<tr>
<td>delay_seconds</td>
<td>Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if <code>PUSH</code> is called from a tight loop.</td>
</tr>
<tr>
<td>transaction_count</td>
<td>If ( &gt;0 ), then the maximum number of transactions to be pushed before stopping. If <code>transaction_count</code> and <code>execution_seconds</code> are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.</td>
</tr>
</tbody>
</table>
SET_DISABLED Procedure

To disable or enable 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 12–28  SET_DISABLED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>The fully qualified database name of the node whose propagation status you want to change.</td>
</tr>
<tr>
<td>disabled</td>
<td>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.</td>
</tr>
<tr>
<td>catchup</td>
<td>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.</td>
</tr>
<tr>
<td>override</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 12–29  SET_DISABLED Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>No entry was found in the DEFSCHEDULE view for the specified destination.</td>
</tr>
<tr>
<td>cantsetdisabled</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 12–30  UNREGISTER_PROPAGATOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the propagator user.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout in seconds. If the propagator is in use, then the procedure waits until timeout. The default is DBMS_LOCK.MAXWAIT.</td>
</tr>
</tbody>
</table>

Exceptions

Table 12–31  UNREGISTER_PROPAGATOR Procedure Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingpropagator</td>
<td>Specified user is not a propagator.</td>
</tr>
<tr>
<td>propagator_inuse</td>
<td>Propagator is in use, and thus cannot be unregistered. Try later.</td>
</tr>
</tbody>
</table>
### 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();
```

**Parameters**

None

### 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 12–32 UNSCHEDULE_PUSH Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>Fully qualified path name for the database at which you want to unschedule periodic execution of deferred remote procedure calls.</td>
</tr>
</tbody>
</table>

**Table 12–33 UNSCHEDULE_PUSH Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA_FOUND</td>
<td>No entry was found in the DEFSCHEDULE view for the specified dblink.</td>
</tr>
</tbody>
</table>
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:

```plsql
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>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>ORU 10035: cannot describe a package ('X') only a procedure within a package.</td>
</tr>
<tr>
<td>ORA-20001</td>
<td>ORU-10032: procedure 'X' within package 'Y' does not exist.</td>
</tr>
<tr>
<td>ORA-20002</td>
<td>ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.</td>
</tr>
<tr>
<td>ORA-20003</td>
<td>ORU-10036: object 'X' is invalid and cannot be described.</td>
</tr>
<tr>
<td>ORA-20004</td>
<td>Syntax error attempting to parse 'X'.</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_DESCRIBE Subprograms

Syntax

`DBMS_DESCRIBE.DESCRIPTION_PROCEDURE (`n
  `object_name` IN VARCHAR2, `n
  `reserved1` IN VARCHAR2, `n
  `reserved2` IN VARCHAR2, `n
  `overload` OUT NUMBER_TABLE, `n
  `position` OUT NUMBER_TABLE, `n
  `level` OUT NUMBER_TABLE, `n
  `argument_name` OUT VARCHAR2_TABLE, `n
  `datatype` OUT NUMBER_TABLE, `n
  `default_value` OUT NUMBER_TABLE, `n
  `in_out` OUT NUMBER_TABLE, `n
  `length` OUT NUMBER_TABLE, `n
  `precision` OUT NUMBER_TABLE, `n
  `scale` OUT NUMBER_TABLE, `n
  `radix` OUT NUMBER_TABLE, `n
  `spare` OUT NUMBER_TABLE);`n

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object_name</code></td>
<td>Name of the procedure being described.</td>
</tr>
<tr>
<td></td>
<td>The syntax for this parameter follows the rules used for identifiers in SQL.</td>
</tr>
<tr>
<td></td>
<td>The name can be a synonym. This parameter is required and may not be null.</td>
</tr>
<tr>
<td></td>
<td>The total length of the name cannot exceed 197 bytes. An incorrectly specified <code>OBJECT_NAME</code> can result in one of the following exceptions:</td>
</tr>
<tr>
<td></td>
<td>ORA-20000 - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function.</td>
</tr>
<tr>
<td></td>
<td>ORA-20001 - The procedure or function that you specified does not exist within the given package.</td>
</tr>
<tr>
<td></td>
<td>ORA-20002 - The object that you specified is a remote object. This procedure cannot currently describe remote objects.</td>
</tr>
<tr>
<td></td>
<td>ORA-20003 - The object that you specified is invalid and cannot be described.</td>
</tr>
<tr>
<td></td>
<td>ORA-20004 - The object was specified with a syntax error.</td>
</tr>
<tr>
<td><code>reserved1</code></td>
<td>Reserved for future use -- must be set to NULL or the empty string.</td>
</tr>
<tr>
<td><code>reserved2</code></td>
<td></td>
</tr>
</tbody>
</table>
### Table 13–2  DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overload</td>
<td>A unique number assigned to the procedure’s signature.</td>
</tr>
<tr>
<td></td>
<td>If a procedure is overloaded, then this field holds a different value for</td>
</tr>
<tr>
<td></td>
<td>each version of the procedure.</td>
</tr>
<tr>
<td>position</td>
<td>Position of the argument in the parameter list.</td>
</tr>
<tr>
<td></td>
<td>Position 0 returns the values for the return type of a function.</td>
</tr>
<tr>
<td>level</td>
<td>If the argument is a composite type, such as record, then this parameter</td>
</tr>
<tr>
<td></td>
<td>returns the level of the datatype.</td>
</tr>
<tr>
<td></td>
<td>See the Oracle Call Interface Programmer’s Guide for a description of the</td>
</tr>
<tr>
<td></td>
<td>ODESSP call for an example.</td>
</tr>
<tr>
<td>argument_name</td>
<td>Name of the argument associated with the procedure that you are describing.</td>
</tr>
<tr>
<td>datatype</td>
<td>Oracle datatype of the argument being described.</td>
</tr>
<tr>
<td></td>
<td>The datatypes and their numeric type codes are:</td>
</tr>
<tr>
<td></td>
<td>0  placeholder for procedures with no arguments</td>
</tr>
<tr>
<td></td>
<td>1  VARCHAR, VARCHAR, STRING</td>
</tr>
<tr>
<td></td>
<td>2  NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL</td>
</tr>
<tr>
<td></td>
<td>3  BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL</td>
</tr>
<tr>
<td></td>
<td>8  LONG</td>
</tr>
<tr>
<td></td>
<td>11 ROWID</td>
</tr>
<tr>
<td></td>
<td>12 DATE</td>
</tr>
<tr>
<td></td>
<td>23 RAW</td>
</tr>
<tr>
<td></td>
<td>24 LONG RAW</td>
</tr>
<tr>
<td></td>
<td>96 CHAR (ANSI FIXED CHAR), CHARACTER</td>
</tr>
<tr>
<td></td>
<td>106 MLSLABEL</td>
</tr>
<tr>
<td></td>
<td>250 PL/SQL RECORD</td>
</tr>
<tr>
<td></td>
<td>251 PL/SQL TABLE</td>
</tr>
<tr>
<td></td>
<td>252 PL/SQL BOOLEAN</td>
</tr>
<tr>
<td>default_value</td>
<td>1 if the argument being described has a default value; otherwise, the</td>
</tr>
<tr>
<td></td>
<td>value is 0.</td>
</tr>
<tr>
<td>in_out</td>
<td>Describes the mode of the parameter:</td>
</tr>
<tr>
<td></td>
<td>0 IN</td>
</tr>
<tr>
<td></td>
<td>1 OUT</td>
</tr>
<tr>
<td></td>
<td>2 IN IN</td>
</tr>
<tr>
<td>length</td>
<td>Data length, in bytes, of the argument being described.</td>
</tr>
<tr>
<td>precision</td>
<td>If the argument being described is of datatype 2 (NUMBER), then this</td>
</tr>
<tr>
<td></td>
<td>parameter is the precision of that number.</td>
</tr>
</tbody>
</table>
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.

Examples

One use of the `DESCRIBE_PROCEDURE` procedure would be 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;
```

The describe of this procedure might look similar to the output shown below.

<table>
<thead>
<tr>
<th>overload</th>
<th>position</th>
<th>argument</th>
<th>level</th>
<th>datatype</th>
<th>length</th>
<th>prec</th>
<th>scale</th>
<th>rad</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>22</td>
<td>7</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```sql
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,  
pplisint 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,  
pnume IN NUMERIC,  
pdip IN DOUBLE PRECISION,  
pdate IN DATE,  
pmls IN MLSLABEL) AS
```
BEGIN
   NULL;
END;

If you describe this procedure using the package below:

CREATE OR REPLACE PACKAGE describe_it AS
   PROCEDURE descproc (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;
   END prt_value;

   PROCEDURE descproc (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 13-7
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, as shown below, list all the numeric codes for the PL/SQL datatypes:

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Datatype_Code</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PVC2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>PVC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>PSTR</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>PLONG</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>PROWID</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>PCHARA</td>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>PCHAR</td>
<td>96</td>
<td>0</td>
</tr>
</tbody>
</table>
Summary of DBMS_DESCRIBE Subprograms

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 PDF 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 DTYPES (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 last example above (see function H).
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 OEM.

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
- Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ALLOW_ALL Procedure&quot; on page 14-2</td>
<td>Empties the list and inserts a row indicating that all servers should be trusted.</td>
</tr>
<tr>
<td>&quot;ALLOW_SERVER Procedure&quot; on page 14-3</td>
<td>Enables a specific server to be allowed access even though deny all is indicated in the list.</td>
</tr>
<tr>
<td>&quot;DENY_ALL Procedure&quot; on page 14-3</td>
<td>Empties the list and inserts a row indicating that all servers should be untrusted.</td>
</tr>
<tr>
<td>&quot;DENY_SERVER Procedure&quot; on page 14-4</td>
<td>Enables a specific server to be denied access even though allow all is indicated in the list.</td>
</tr>
</tbody>
</table>

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.
ALLOW_SERVER Procedure

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 14–2  ALLOW_SERVER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Unique, fully-qualified name of the server to be trusted.</td>
</tr>
</tbody>
</table>

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.
**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Unique, fully-qualified name of the server to be untrusted.</td>
</tr>
</tbody>
</table>

**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:

```sql
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER
    ('SALES.US.AMERICAS.ACME_AUTO.COM');
```

Statement processed.

```
SELECT * FROM TRUSTED_SERVERS;
```

<table>
<thead>
<tr>
<th>TRUST</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted</td>
<td>SALES.US.AMERICAS.ACME_AUTO.COM</td>
</tr>
</tbody>
</table>

1 row selected.

By executing the **DENY_ALL** procedure, you can choose to not trust any database server:

```sql
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

Statement processed.

```
SELECT * FROM TRUSTED_SERVERS;
```

<table>
<thead>
<tr>
<th>TRUST</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted</td>
<td>All</td>
</tr>
</tbody>
</table>

1 row selected.

The **ALLOW_SERVER** procedure can be used to specify that one particular database is to be trusted:

```sql
EXECUTE
    DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER
    ('SALES.US.AMERICAS.ACME_AUTO.COM');
```

Statement processed.

```
SELECT * FROM TRUSTED_SERVERS;
```

<table>
<thead>
<tr>
<th>TRUST</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted</td>
<td>SALES.US.AMERICAS.ACME_AUTO.COM</td>
</tr>
</tbody>
</table>
Trusted SALES.US.AMERICA.ACM.COM

1 row selected.
DENY_SERVER Procedure
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.

---

**Note:** **DBMS_RLS** is only available with the Enterprise Edition.

This chapter discusses the following topics:

- **Summary of DBMS_FGA Subprogram**
Summary of DBMS_FGA Subprogram

Table 15–1 Summary of DBMS_FGA Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_POLICY Procedure&quot; on page 15-2</td>
<td>Creates an audit policy using the supplied predicate as the audit condition</td>
</tr>
<tr>
<td>&quot;DROP_POLICY Procedure&quot; on page 15-3</td>
<td>Drops an audit policy</td>
</tr>
<tr>
<td>&quot;ENABLE_POLICY Procedure&quot; on page 15-4</td>
<td>Enables an audit policy</td>
</tr>
<tr>
<td>&quot;DISABLE_POLICY Procedure&quot; on page 15-4</td>
<td>Disables an audit policy</td>
</tr>
</tbody>
</table>

ADD_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition.

Syntax

```sql
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 15–2 ADD_POLICY Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema of the object to be audited</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>The unique name of the policy</td>
</tr>
<tr>
<td>audit_condition</td>
<td>A condition in a row that indicates a monitoring condition</td>
</tr>
</tbody>
</table>
Summary of DBMS_FGA Subprogram

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 );
```
**ENABLE_POLICY Procedure**

### Parameters

**Table 15–3 DROP_POLICY Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema of the object to be audited</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>The unique name of the policy</td>
</tr>
</tbody>
</table>

### 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**

```sql
DBMS_FGA.ENABLE_POLICY(
  object_schema VARCHAR2,
  object_name VARCHAR2,
  policy_name VARCHAR2 );
```

**Parameters**

**Table 15–4**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema of the object to be audited</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>The unique name of the policy</td>
</tr>
</tbody>
</table>

**DISABLE_POLICY Procedure**

This procedure disables an audit policy.
Summary of DBMS_FGA Subprogram

Syntax

DBMS_FGA.DISABLE_POLICY(
    object_schema VARCHAR2,
    object_name VARCHAR2,
    policy_name VARCHAR2 );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema of the object to be audited</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>The unique name of the policy</td>
</tr>
</tbody>
</table>
DISABLE_POLICY Procedure
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 for detailed information about DBMS_FLASHBACK.

This chapter discusses the following topics:

- DBMS_FLASHBACK Error Messages
- 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.

```sql
/* keep_scn is a temporary table to store scns that we are interested in. */
create table keep_scn (scn number);
execute dbms_flashback.disable;
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
);

/* 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');
```

### DBMS_FLASHBACK Error Messages

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8182</td>
<td>In Flashback mode, user cannot perform DML or DDL operations.</td>
</tr>
<tr>
<td>8184</td>
<td>User cannot enable Flashback within another Flashback session.</td>
</tr>
<tr>
<td>8183</td>
<td>User cannot enable Flashback within an uncommitted transaction.</td>
</tr>
<tr>
<td>8185</td>
<td>SYS cannot enable Flashback mode. User cannot begin read only or serializable transactions in Flashback mode.</td>
</tr>
<tr>
<td>8180</td>
<td>Time specified is too old.</td>
</tr>
<tr>
<td>8181</td>
<td>Invalid system change number specified.</td>
</tr>
</tbody>
</table>
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;
/* 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;
execute dbms_flashback.disable;

/* 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;
/* Scott decides to retire but the transaction is done incorrectly */
delete from employee where employee_name = 'Scott Tiger';
commit;
/* 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;
/* Put back scott's organization */
declare
restore_scn date;
begi
select scn into restore_scn from keep_scn;
dbms_flashback.enable_at_system_change_number (restore_scn);
end;
/* 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;

declare
scotts_emp number;
scotts_mgr number;
cursor cl 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 is cl % ROWTYPE;
begin
  select employee_no, employee_mgr into scotts_emp, scotts_mgr from employee
  where employee_name = 'Scott Tiger';
  /* Open cl with Flashback enabled. */
  open cl;
  /* Disable Flashback. */
  dbms_flashback.disable;
  loop
    /* Note that all the DML operations inside the loop are performed with
     Flashback disabled. */
    fetch c1 into c1_rec;
    exit when c1%NOTFOUND;
    for c1_rec in c1 loop
      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;
/
Summary of DBMS_FLASHBACK Subprograms

end if;
end loop;
end;
/

execute dbms_flashback. disable;

---

**Summary of DBMS_FLASHBACK Subprograms**

<table>
<thead>
<tr>
<th><strong>Table 16–2  DBMS_FLASHBACK Subprograms</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subprogram</strong></td>
</tr>
<tr>
<td>&quot;ENABLE_AT_TIME Procedure&quot; on page 16-6</td>
</tr>
<tr>
<td>&quot;ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure&quot; on page 16-7</td>
</tr>
<tr>
<td>&quot;GET_SYSTEMCHANGE_NUMBER Function&quot; on page 16-8</td>
</tr>
<tr>
<td>&quot;DISABLE Procedure&quot; on page 16-8</td>
</tr>
</tbody>
</table>

---

**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**

```sql
DBMS_FLASHBACK.ENABLE_AT_TIME (
    query_time IN TIMESTAMP);
```

**Parameters**

Table 16–3 shows the parameters for the ENABLE_AT_TIME procedure.
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

```sql
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE (  
    query_scn IN NUMBER);
```

Parameters

Table 16–4 shows the parameters for the ENABLE_AT_SYSTEM_CHANGE_NUMBER procedure.
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;
```
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.

See Also: For detailed information on Heterogeneous Services and on binding variables, see Oracle9i Distributed Database Systems.

This chapter discusses the following topics:

- Security
- Summary of DBMS_HS_PASSTHROUGH Subprograms
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BIND_VARIABLE Procedure&quot; on page 17-3</td>
<td>Binds an IN variable positionally with a PL/SQL program variable.</td>
</tr>
<tr>
<td>&quot;BIND_VARIABLE_RAW Procedure&quot; on page 17-4</td>
<td>Binds IN variables of type RAW.</td>
</tr>
<tr>
<td>&quot;BIND_OUT_VARIABLE Procedure&quot; on page 17-5</td>
<td>Binds an OUT variable with a PL/SQL program variable.</td>
</tr>
<tr>
<td>&quot;BIND_OUT_VARIABLE_RAW Procedure&quot; on page 17-7</td>
<td>Binds an OUT variable of datatype RAW with a PL/SQL program variable.</td>
</tr>
<tr>
<td>&quot;BIND_INOUT_VARIABLE Procedure&quot; on page 17-8</td>
<td>Binds IN OUT bind variables.</td>
</tr>
<tr>
<td>&quot;BIND_INOUT_VARIABLE_RAW Procedure&quot; on page 17-9</td>
<td>Binds IN OUT bind variables of datatype RAW.</td>
</tr>
<tr>
<td>&quot;CLOSE_CURSOR Procedure&quot; on page 17-10</td>
<td>Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system.</td>
</tr>
<tr>
<td>&quot;EXECUTE_IMMEDIATE Procedure&quot; on page 17-11</td>
<td>Runs a (non-SELECT) SQL statement immediately, without bind variables.</td>
</tr>
<tr>
<td>&quot;EXECUTE_NON_QUERY Function&quot; on page 17-12</td>
<td>Runs a (non-SELECT) SQL statement.</td>
</tr>
<tr>
<td>&quot;FETCH_ROW Function&quot; on page 17-13</td>
<td>Fetches rows from a query.</td>
</tr>
<tr>
<td>&quot;GET_VALUE Procedure&quot; on page 17-14</td>
<td>Retrieves column value from SELECT statement, or retrieves OUT bind parameters.</td>
</tr>
<tr>
<td>&quot;GET_VALUE_RAW Procedure&quot; on page 17-15</td>
<td>Similar to GET_VALUE, but for datatype RAW.</td>
</tr>
<tr>
<td>&quot;OPEN_CURSOR Function&quot; on page 17-16</td>
<td>Opens a cursor for running a passthrough SQL statement at the non-Oracle system.</td>
</tr>
</tbody>
</table>
**BIND_VARIABLE Procedure**

This procedure binds an `IN` variable positionally with a PL/SQL program variable.

**Syntax**

```sql
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 17-4.

**Parameters**

**Table 17–2 BIND_VARIABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines <code>OPEN_CURSOR</code> and <code>PARSE</code> respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Value that must be passed to the bind variable name.</td>
</tr>
<tr>
<td>name</td>
<td>(Optional) Name of the bind variable. For example, in <code>SELECT * FROM emp WHERE ename=:ename</code>, the position of the bind variable <code>ename</code> is 1, the name is <code>ename</code>. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

**Pragmas**

Purity level defined: WNDS, RNDS

**BIND_VARIABLE_RAW Procedure**

This procedure binds IN variables of type RAW.

**Syntax**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Value that must be passed to the bind variable.</td>
</tr>
</tbody>
</table>
Summary of DBMS_HS_PASSTHROUGH Subprograms

Exceptions

Table 17–5 BIND_VARIABLE_RAW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined: WNDS, RNDS

BIND_OUT_VARIABLE Procedure

This procedure binds an OUT variable with a PL/SQL program variable.

Syntax

```sql
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 17-7.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Variable in which the OUT bind variable stores its value. The package remembers only the &quot;size&quot; 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.</td>
</tr>
<tr>
<td>name</td>
<td>(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 &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (  
c IN BINARY_INTEGER NOT NULL,  
pos IN BINARY_INTEGER NOT NULL,  
val OUT RAW,  
name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Variable in which the OUT bind variable stores its value. The package remembers only the &quot;size&quot; 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.</td>
</tr>
<tr>
<td>name</td>
<td>(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 &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
</tbody>
</table>
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 17-9.

Parameters

**Table 17–10  BIND_INOUT_VARIABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
</tbody>
</table>
| 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. |

Pragmas

Purity level defined : WNDS, RNdS
Table 17–10  BIND_INOUT_VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>(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 &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

Table 17–11  BIND_INOUT_VARIABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined : WNDS, RNDS

BIND_INOUT_VARIABLE_RAW Procedure

This procedure binds IN OUT bind variables of datatype RAW.

Syntax

```sql
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);
```
CLOSE_CURSOR Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>(Optional) Name the bind variable. For example, in <code>SELECT * FROM emp WHERE ename=:ename</code>, the position of the bind variable <code>:ename</code> is 1, the name is <code>:ename</code>. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
   c IN BINARY_INTEGER NOT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor to be released.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined : WNGS, 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

```sql
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
   s IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```
EXECUTE_NON_QUERY Function

Parameters

<table>
<thead>
<tr>
<th>Table 17–16 EXECUTE_IMMEDIATE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>s</td>
</tr>
</tbody>
</table>

Returns

The number of rows affected by the execution of the SQL statement.

Exceptions

<table>
<thead>
<tr>
<th>Table 17–17 EXECUTE_IMMEDIATE Procedure Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
</tr>
<tr>
<td>ORA-28551</td>
</tr>
<tr>
<td>ORA-28544</td>
</tr>
<tr>
<td>ORA-28555</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY ( 
    c IN BINARY_INTEGER NOT NULL) 
RETURN BINARY_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Table 17–18 EXECUTE_NON_QUERY Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>
Summary of DBMS_HS_PASSTHROUGH Subprograms

Returns

The number of rows affected by the SQL statement in the non-Oracle system.

Exceptions

Table 17–19  EXECUTE_NON_QUERY Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>BIND_VARIABLE procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

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 17–20  FETCH_ROW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
</tbody>
</table>
| 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). |
GET_VALUE Procedure

Returns

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 17–21  FETCH_ROW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

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 17-15.
Summary of DBMS_HS_P ASSTHROUGH Subprograms

Parameters

Table 17–22  GET_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable or select list item in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Variable in which the OUT bind variable or select list item stores its value.</td>
</tr>
</tbody>
</table>

Exceptions

Table 17–23  GET_VALUE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-1403</td>
<td>Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (i.e., FETCH_ROW returned &quot;0&quot;).</td>
</tr>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined : WNDS

GET_VALUE_RAW Procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```sql
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
    c IN BINARY_INTEGER NOT NULL,
    pos IN BINARY_INTEGER NOT NULL,
    val OUT RAW);
```
OPEN_CURSOR Function

Parameters

Table 17–24  GET_VALUE_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the bind variable or select list item in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>val</td>
<td>Variable in which the OUT bind variable or select list item stores its value.</td>
</tr>
</tbody>
</table>

Exceptions

Table 17–25  GET_VALUE_RAW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA–1403</td>
<td>Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (i.e., FETCH_ROW returned &quot;0&quot;).</td>
</tr>
<tr>
<td>ORA–28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA–28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA–28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA–28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_HS_P ASSTHROUGH Subprograms

Syntax

```sql
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
RETURN BINARY_INTEGER;
```

Returns

The cursor to be used on subsequent procedure and function calls.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28554</td>
<td>Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' OPEN_CURSORS initialization parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined: WNDS, RNDS

PARSE Procedure

This procedure parses SQL statement at non-Oracle system.

Syntax

```sql
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (  
c IN BINARY_INTEGER NOT NULL,  
stmt IN VARCHAR2 NOT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the pass-through SQL statement. Cursor must be opened using function OPEN_CURSOR.</td>
</tr>
<tr>
<td>stmt</td>
<td>Statement to be parsed.</td>
</tr>
</tbody>
</table>
Exceptions

Table 17–28  GET_VALUE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28551</td>
<td>SQL statement is illegal.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

Pragmas

Purity level defined: WNDS, RNDS
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. It 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.

This chapter discusses the following topics:

- **Summary of DBMS_IOT Subprograms**
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**

```sql
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (  
    owner IN VARCHAR2,  
    iot_name IN VARCHAR2,  
    chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>owner</code></td>
<td>Owner of the index-organized table.</td>
</tr>
<tr>
<td><code>iot_name</code></td>
<td>Index-organized table name.</td>
</tr>
<tr>
<td><code>chainrow_table_name</code></td>
<td>Intended name for the chained-rows table.</td>
</tr>
</tbody>
</table>

**Example**

```sql
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:
### 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the index-organized table.</td>
</tr>
<tr>
<td>iot_name</td>
<td>Index-organized table name.</td>
</tr>
<tr>
<td>exceptions_table_name</td>
<td>Intended name for exception-table.</td>
</tr>
</tbody>
</table>
Example

EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS','L','LE');

An exception table for the above index-organized table with the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW_ID</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>OWNER</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>CHAR (16)</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>CHAR (16)</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>CHAR (16)</td>
</tr>
</tbody>
</table>
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:

```sql
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)
```

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:

```sql
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:

```sql
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.

```sql
DBMS_JOB.RUN( JOB IN BINARY_INTEGER,
               FORCE IN BOOLEAN DEFAULT FALSE)
```

**See Also:** For more information about Oracle Real Application Clusters, please refer to Oracle9i Real Application Clusters Concepts.

### Summary of DBMS_JOB Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SUBMIT Procedure&quot; on page 19-4</td>
<td>Submits a new job to the job queue.</td>
</tr>
</tbody>
</table>
Table 19–1  DBMS_JOB Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;REMOVE Procedure&quot; on page 19-6</td>
<td>Removes specified job from the job queue.</td>
</tr>
<tr>
<td>&quot;CHANGE Procedure&quot; on page 19-6</td>
<td>Alters any of the user-definable parameters associated with a job.</td>
</tr>
<tr>
<td>&quot;WHAT Procedure&quot; on page 19-7</td>
<td>Alters the job description for a specified job.</td>
</tr>
<tr>
<td>&quot;NEXT_DATE Procedure&quot; on page 19-8</td>
<td>Alters the next execution time for a specified job.</td>
</tr>
<tr>
<td>&quot;INSTANCE Procedure&quot; on page 19-8</td>
<td>Assigns a job to be run by a instance.</td>
</tr>
<tr>
<td>&quot;INTERVAL Procedure&quot; on page 19-9</td>
<td>Alters the interval between executions for a specified job.</td>
</tr>
<tr>
<td>&quot;BROKEN Procedure&quot; on page 19-10</td>
<td>Disables job execution.</td>
</tr>
<tr>
<td>&quot;RUN Procedure&quot; on page 19-11</td>
<td>Forces a specified job to run.</td>
</tr>
<tr>
<td>&quot;USER_EXPORT Procedure&quot; on page 19-11</td>
<td>Recreates a given job for export.</td>
</tr>
<tr>
<td>&quot;USER_EXPORT Procedure&quot; on page 19-12</td>
<td>Recreates a given job for export with instance affinity.</td>
</tr>
</tbody>
</table>

**SUBMIT Procedure**

This procedure submits a new job. It chooses the job from the sequence `sys.jobseq`.

**Syntax**

```sql
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 19–2  SUBMIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL procedure to run.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date when the job will be run.</td>
</tr>
<tr>
<td>interval</td>
<td>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.</td>
</tr>
<tr>
<td>no_parse</td>
<td>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.</td>
</tr>
<tr>
<td>instance</td>
<td>When a job is submitted, specifies which instance can run the job.</td>
</tr>
<tr>
<td>force</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

```sql
VARIABLE jobno number;
BEGIN
    DBMS_JOB.SUBMIT(:jobno,
        'dbms_ddl.analyze_object(''TABLE'',
        ''DQUON'', ''ACCOUNTS''),
```
REMOVE Procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

Syntax

```sql
DBMS_JOB.REMOVE (
    job IN BINARY_INTEGER );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
</tbody>
</table>

Example

```sql
EXECUTE DBMS_JOB.REMOVE(14144);
```

CHANGE Procedure

This procedure changes any of the user-settable fields in a job.

Syntax

```sql
DBMS_JOB.CHANGE (
    job IN BINARY_INTEGER,
    what IN VARCHAR2,
    next_date IN DATE,
    interval IN VARCHAR2,
    instance IN BINARY_INTEGER DEFAULT NULL,
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL procedure to run.</td>
</tr>
<tr>
<td>next_date</td>
<td>Date of the next refresh.</td>
</tr>
<tr>
<td>interval</td>
<td>Date function; evaluated immediately before the job starts running.</td>
</tr>
<tr>
<td>instance</td>
<td>When a job is submitted, specifies which instance can run the job. This defaults to NULL, which indicates that instance affinity is not changed.</td>
</tr>
<tr>
<td>force</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```sql
EXECUTE DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
```

WHAT Procedure

This procedure changes what an existing job does, and replaces its environment.

Syntax

```sql
DBMS_JOB.WHAT (  
    job       IN BINARY_INTEGER,  
```

```sql
```
Parameters

Table 19–5 WHAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL procedure to run.</td>
</tr>
</tbody>
</table>

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 19–6 NEXT_DATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>next_date</td>
<td>Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.</td>
</tr>
</tbody>
</table>

INSTANCE Procedure

This procedure changes job instance affinity.
Summary of DBMS_JOB Subprograms

Syntax

```sql
DBMS_JOB.INSTANCE (  
    job     IN BINARY_INTEGER,  
    instance IN BINARY_INTEGER,  
    force   IN BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>instance</td>
<td>When a job is submitted, a user can specify which instance can run the job.</td>
</tr>
<tr>
<td>force</td>
<td>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.</td>
</tr>
</tbody>
</table>

**INTERVAL Procedure**

This procedure changes how often a job runs.

Syntax

```sql
DBMS_JOB.INTERVAL (  
    job     IN BINARY_INTEGER,  
    interval IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>interval</td>
<td>Date function, evaluated immediately before the job starts running.</td>
</tr>
</tbody>
</table>
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:

- `'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

```sql
DBMS_JOB.BROKEN (    job IN BINARY_INTEGER,    broken IN BOOLEAN,    next_date IN DATE DEFAULT SYSDATE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>job</code></td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td><code>broken</code></td>
<td>Job broken: IN value is FALSE.</td>
</tr>
<tr>
<td><code>next_date</code></td>
<td>Date of the next refresh.</td>
</tr>
</tbody>
</table>

**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

```sql
DBMS_JOB.RUN (
    job IN BINARY_INTEGER,
    force IN BOOLEAN DEFAULT FALSE);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>force</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### Example

```sql
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 recreate the given job.

#### Syntax

```sql
DBMS_JOB.USER_EXPORT (
    job IN BINARY_INTEGER,
    mycall IN OUT VARCHAR2);
```
USER_EXPORT Procedure

Parameters

Table 19–11 USER_EXPORT Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>mycall</td>
<td>Text of a call to recreate the given job.</td>
</tr>
</tbody>
</table>

USER_EXPORT Procedure

This procedure alters instance affinity (8i and above) and preserves the compatibility.

Syntax

DBMS_JOB.USER_EXPORT (  
  job IN BINARY_INTEGER,  
  mycall IN OUT VARCHAR2,  
  myinst IN OUT VARCHAR2);  

Parameters

Table 19–12 USER_EXPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>Number of the job being run.</td>
</tr>
<tr>
<td>mycall</td>
<td>Text of a call to recreate a given job.</td>
</tr>
<tr>
<td>myinst</td>
<td>Text of a call to alter instance affinity.</td>
</tr>
</tbody>
</table>
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
- Summary of Data Types
- Summary of DBMS_LDAP Subprograms
Table 20–1 lists the exceptions generated by DBMS_LDAP.

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Oracle Error</th>
<th>Cause of Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_error</td>
<td>31202</td>
<td>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.</td>
</tr>
<tr>
<td>init_failed</td>
<td>31203</td>
<td>Raised by DBMS_LDAP.init if there are some problems.</td>
</tr>
<tr>
<td>invalid_session</td>
<td>31204</td>
<td>Raised by all functions and procedures in the DBMS_LDAP package if they are passed an invalid session handle.</td>
</tr>
<tr>
<td>invalid_auth_method</td>
<td>31205</td>
<td>Raised by DBMS_LDAP.bind_s if the authentication method requested is not supported.</td>
</tr>
<tr>
<td>invalid_search_scope</td>
<td>31206</td>
<td>Raised by all of the search functions if the scope of the search is invalid.</td>
</tr>
<tr>
<td>invalid_search_time_val</td>
<td>31207</td>
<td>Raised by time based search function: DBMS_LDAP.search_st if it is given an invalid value for the time limit.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>31208</td>
<td>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.</td>
</tr>
<tr>
<td>count_entry_error</td>
<td>31209</td>
<td>Raised by DBMS_LDAP.count_entries if it cannot count the entries in a given result set.</td>
</tr>
<tr>
<td>get_dn_error</td>
<td>31210</td>
<td>Raised by DBMS_LDAP.get_dn if the DN of the entry it is retrieving is NULL.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>31211</td>
<td>Raised by all the functions that modify/add/rename an entry if they are presented with an invalid entry DN.</td>
</tr>
<tr>
<td>invalid_mod_array</td>
<td>31212</td>
<td>Raised by all functions that take a modification array as an argument if they are given an invalid modification array.</td>
</tr>
<tr>
<td>invalid_mod_option</td>
<td>31213</td>
<td>Raised by DBMS_LDAP.populate_mod_array if the modification option given is anything other than MOD_ADD, MOD_DELETE or MOD_REPLACE.</td>
</tr>
<tr>
<td>invalid_mod_type</td>
<td>31214</td>
<td>Raised by DBMS_LDAP.populate_mod_array if the attribute type that is being modified is NULL.</td>
</tr>
<tr>
<td>invalid_mod_value</td>
<td>31215</td>
<td>Raised by DBMS_LDAP.populate_mod_array if the modification value parameter for a given attribute is NULL.</td>
</tr>
<tr>
<td>invalid_rdn</td>
<td>31216</td>
<td>Raised by all functions and procedures that expect a valid RDN if the value of the RDN is NULL.</td>
</tr>
</tbody>
</table>
Summary of Data Types

The DBMS_LDAP package uses the data types shown in Table 20–2.

Table 20–2  DBMS_LDAP Summary of Data Types

<table>
<thead>
<tr>
<th>Data-Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION</td>
<td>Holds the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Holds a handle to the message retrieved from the result set. This is used by all functions that work with entries, attributes, and values.</td>
</tr>
<tr>
<td>MOD_ARRAY</td>
<td>Holds a handle into the array of modifications being passed into either modify_s or add_s.</td>
</tr>
<tr>
<td>TIMEVAL</td>
<td>Passes time limit information to the LDAP API functions that require a time limit.</td>
</tr>
<tr>
<td>BER_ELEMENT</td>
<td>Holds a handle to a BER structure used for decoding incoming messages.</td>
</tr>
</tbody>
</table>
Summary of DBMS_LDAP Subprograms

Table 20–2  DBMS_LDAP Summary of Data Types

<table>
<thead>
<tr>
<th>Data-Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING_COLLECTION</td>
<td>Holds a list of VARCHAR2 strings which can be passed on to the LDAP server.</td>
</tr>
<tr>
<td>BINVAL_COLLECTION</td>
<td>Holds a list of RAW data which represent binary data.</td>
</tr>
<tr>
<td>BERVAL_COLLECTION</td>
<td>Holds a list of BERVAL values that are used for populating a modification array.</td>
</tr>
</tbody>
</table>

Summary of DBMS_LDAP Subprograms

Table 20–3  DBMS_LDAP Subprograms

<table>
<thead>
<tr>
<th>Function or Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;init Function&quot; on page 20-6</td>
<td>Initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.</td>
</tr>
<tr>
<td>&quot;simple_bind_s Function&quot; on page 20-8</td>
<td>Performs simple username/password based authentication to the directory server.</td>
</tr>
<tr>
<td>&quot;bind_s Function&quot; on page 20-9</td>
<td>Performs complex authentication to the directory server.</td>
</tr>
<tr>
<td>&quot;unbind_s Function&quot; on page 20-12</td>
<td>Closes an active LDAP session.</td>
</tr>
<tr>
<td>&quot;compare_s Function&quot; on page 20-13</td>
<td>Tests if a particular attribute in a particular entry has a particular value.</td>
</tr>
<tr>
<td>&quot;search_s Function&quot; on page 20-15</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;search_st Function&quot; on page 20-17</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;first_entry Function&quot; on page 20-20</td>
<td>Retrieves the first entry in the result set returned by either search_s or search_st.</td>
</tr>
<tr>
<td>&quot;next_entry Function&quot; on page 20-21</td>
<td>Iterates to the next entry in the result set of a search operation.</td>
</tr>
</tbody>
</table>
Table 20–3 DBMS_LDAP Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Function or Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;count_entries Function&quot; on page 20-23</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;first_attribute Function&quot; on page 20-25</td>
<td>Fetches the first attribute of a given entry in the result set.</td>
</tr>
<tr>
<td>&quot;next_attribute Function&quot; on page 20-27</td>
<td>Fetches the next attribute of a given entry in the result set.</td>
</tr>
<tr>
<td>&quot;get_dn Function&quot; on page 20-29</td>
<td>Retrieves the X.500 distinguished name of a given entry in the result set.</td>
</tr>
<tr>
<td>&quot;get_values Function&quot; on page 20-30</td>
<td>Retrieves all of the values associated for a given attribute in a given entry.</td>
</tr>
<tr>
<td>&quot;get_values_len Function&quot; on page 20-32</td>
<td>Retrieves values of attributes that have a Binary syntax.</td>
</tr>
<tr>
<td>&quot;delete_s Function&quot; on page 20-34</td>
<td>Removes a leaf entry in the LDAP Directory Information Tree.</td>
</tr>
<tr>
<td>&quot;modrdn2_s Function&quot; on page 20-35</td>
<td>Renames the relative distinguished name of an entry.</td>
</tr>
<tr>
<td>&quot;err2string Function&quot; on page 20-37</td>
<td>Converts an LDAP error code to string in the local language in which the API is operating.</td>
</tr>
<tr>
<td>&quot;create_mod_array Function&quot; on page 20-38</td>
<td>Allocates memory for array modification entries that are applied to an entry using the modify_s functions.</td>
</tr>
<tr>
<td>&quot;populate_mod_array (String Version) Procedure&quot; on page 20-39</td>
<td>Populates one set of attribute information for add or modify operations.</td>
</tr>
<tr>
<td>&quot;populate_mod_array (Binary Version) Procedure&quot; on page 20-41</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;modify_s Function&quot; on page 20-43</td>
<td>Performs a synchronous modification of an existing LDAP directory entry.</td>
</tr>
<tr>
<td>&quot;add_s Function&quot; on page 20-45</td>
<td>Adds a new entry to the LDAP directory synchronously. Before calling add_s, we have to call DBMS_LDAP.create_mod_array and DBMS_LDAP.populate_mod_array first.</td>
</tr>
<tr>
<td>&quot;free_mod_array Procedure&quot; on page 20-47</td>
<td>Frees the memory allocated by DBMS_LDAP.create_mod_array.</td>
</tr>
</tbody>
</table>
init Function

This function initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.

Syntax

```sql
DBMS_LDAP.init (
    hostname    IN VARCHAR2,
    portnum     IN PLS_INTEGER )
RETURN SESSION;
```

---

**Table 20–3 DBMS_LDAP Subprograms (Cont.)**

<table>
<thead>
<tr>
<th>Function or Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;count_values Function&quot; on page 20-48</td>
<td>Counts the number of values returned by DBMS_LDAP.get_values.</td>
</tr>
<tr>
<td>&quot;count_values_len Function&quot; on page 20-49</td>
<td>Counts the number of values returned by DBMS_LDAP.get_values_len.</td>
</tr>
<tr>
<td>&quot;rename_s Function&quot; on page 20-50</td>
<td>Renames an LDAP entry synchronously.</td>
</tr>
<tr>
<td>&quot;explode_dn Function&quot; on page 20-52</td>
<td>Breaks a DN up into its components.</td>
</tr>
<tr>
<td>&quot;open_ssl Function&quot; on page 20-52</td>
<td>Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.</td>
</tr>
</tbody>
</table>
Parameters

Table 20–4 init Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>portnum (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–5 init Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION</td>
<td>A handle to an LDAP session that can be used for further calls into the API.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–6 init Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_failed</td>
<td>Raised when there is a problem contacting the LDAP server.</td>
</tr>
<tr>
<td>ts_mode_not_supported</td>
<td>Raised if DBMS_LDAP.init is invoked from a user session that is logged onto the database using an MTS service.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with the exception describes the error in detail.</td>
</tr>
</tbody>
</table>

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.
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;
```

See Also:

- "simple_bind_s Function" on page 20-8
- "bind_s Function" on page 20-9
Parameters

Table 20–7 simple_bind_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>dn (IN)</td>
<td>The distinguished name of the user under which you are trying to login.</td>
</tr>
<tr>
<td>passwd (IN)</td>
<td>A text string containing the password.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–8 simple_bind_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP SUCCESS on a successful completion. If there was a problem, one of the exceptions in Table 20–9 is raised.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–9 simple_bind_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>mts_mode_not_supported</td>
<td>Raised if DBMS_LDAP.init is invoked from a user session that is logged onto as an MTS service.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_LDAP.bind_s (
```
bind_s Function

ld IN SESSION,
dn IN VARCHAR2,
passwd IN VARCHAR2,
meth IN PLS_INTEGER )
RETURN PLS_INTEGER;
Parameters

Table 20–10 bind_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>dn</td>
<td>The distinguished name of the user under which you are trying to login.</td>
</tr>
<tr>
<td>cred</td>
<td>A text string containing the credentials used for authentication.</td>
</tr>
<tr>
<td>meth</td>
<td>The authentication method.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–11 bind_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP.SUCCESS on a successful completion. One of the exceptions in Table 20–12 is raised if there is a problem.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–12 bind_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_auth_method</td>
<td>Raised if the authentication method requested is not supported.</td>
</tr>
<tr>
<td>mts_mode_not_supported</td>
<td>Raised if invoked from a user session that is logged onto an MTS service.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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.
unbind_s Function

This function closes an active LDAP session.

Syntax

```sql
DBMS_LDAP.unbind_s ( 
    ld IN SESSION )
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>SUCCESS on proper completion. One of the exceptions listed in Table 20–15 is raised otherwise.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>general error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 20-8
- "bind_s Function" on page 20-9

**compare_s Function**

This function tests whether a particular attribute in a particular entry has a particular value.

**Syntax**

```sql
DBMS_LDAP.compare_s (  
    ld IN SESSION,  
    dn IN VARCHAR2,  
    attr IN VARCHAR2,  
    value IN VARCHAR2)  
RETURN PLS_INTEGER;
```
compare_s Function

Parameters

Table 20–16  compare_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle</td>
</tr>
<tr>
<td>dn (IN)</td>
<td>The name of the entry to compare against</td>
</tr>
<tr>
<td>attr (IN)</td>
<td>The attribute to compare against.</td>
</tr>
<tr>
<td>value (IN)</td>
<td>A string attribute value to compare against</td>
</tr>
</tbody>
</table>

Return Values

Table 20–17  compare_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>COMPARE_TRUE is the given attribute that has a matching value.</td>
</tr>
<tr>
<td></td>
<td>COMPARE_FALSE if the value of the attribute does not match the value given.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–18  compare_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 20-9.

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,  
  attr  IN STRING_COLLECTION,  
  attronly  IN PLS_INTEGER,  
  res  OUT MESSAGE)  
RETURN PLS_INTEGER;
search_s Function

Parameters

Table 20–19 search_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>base (IN)</td>
<td>The dn of the entry at which to start the search.</td>
</tr>
<tr>
<td>scope (IN)</td>
<td>One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.</td>
</tr>
<tr>
<td>filter (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>attrs (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>attrsonly (IN)</td>
<td>A boolean value that must be zero if both attribute types and values are returned, and non-zero if only types are wanted.</td>
</tr>
<tr>
<td>res (OUT)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–20 search_s Function Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.</td>
</tr>
<tr>
<td>res (OUT parameter)</td>
<td>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.</td>
</tr>
</tbody>
</table>
Exceptions

<table>
<thead>
<tr>
<th>Table 20–21</th>
<th>search_s Function Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_search_scope</td>
<td>Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL, or SCOPE_SUBTREE.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 below.

See Also:
- "search_st Function" on page 20-17
- "first_entry Function" on page 20-20
- "next_entry Function" on page 20-21

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)
```
search_st Function

RETURN PLS_INTEGER;
### Parameters

**Table 20–22  search_st Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>base (IN)</td>
<td>The dn of the entry at which to start the search.</td>
</tr>
<tr>
<td>scope (IN)</td>
<td>One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.</td>
</tr>
<tr>
<td>filter (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>attrs (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>attrsonly (IN)</td>
<td>A boolean value that must be zero if both attribute types and values are returned, and non-zero if only types are wanted.</td>
</tr>
<tr>
<td>tv (IN)</td>
<td>The timeout value expressed in seconds and microseconds that should be used for this search.</td>
</tr>
<tr>
<td>res (OUT)</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Return Values

**Table 20–23  search_st Function Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.</td>
</tr>
<tr>
<td>res (OUT parameter)</td>
<td>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.</td>
</tr>
</tbody>
</table>
first_entry Function

Exceptions

Table 20–24  search_st Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_search_scope</td>
<td>Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.</td>
</tr>
<tr>
<td>invalid_search_time_value</td>
<td>Raised if the time value specified for the timeout is invalid.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 20-15
- "first_entry Function" on page 20-20
- "next_entry Function" on page 20-21

first_entry Function

This function retrieves the first entry in the result set returned by either search_s or search_st

Syntax

```sql
DBMS_LDAP.first_entry ( 
  ld  IN SESSION, 
  msg  IN MESSAGE )
RETURN MESSAGE;
```
Parameters

Table 20–25  first_entry Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The search result obtained by a call to one of the synchronous search routines.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–26  first_entry Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–27  first_entry Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming msg handle is invalid.</td>
</tr>
</tbody>
</table>

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 20-21
- "search_s Function" on page 20-15
- "search_st Function" on page 20-17

next_entry Function

This function iterates to the next entry in the result set of a search operation.
next_entry Function

Syntax

```sql
DBMS_LDAP.next_entry (
    ld IN SESSION,
    msg IN MESSAGE
) RETURN MESSAGE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The search result, as obtained by a call to one of the synchronous search routines.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle, ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming msg handle is invalid.</td>
</tr>
</tbody>
</table>

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.
**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**

```sql
DBMS_LDAP.count_entries (
    Id   IN SESSION,
    msg  IN MESSAGE )
RETURN PLS_INTEGER;
```
count_entries Function

Parameters

Table 20–31  count_entry Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The search result, as obtained by a call to one of the synchronous search routines</td>
</tr>
</tbody>
</table>

Return Values

Table 20–32  count_entry Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS INTEGER</td>
<td>Non-zero if there are entries in the result set</td>
</tr>
<tr>
<td></td>
<td>-1 if there was a problem.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–33  count_entry Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming msg handle is invalid.</td>
</tr>
<tr>
<td>count_entry_error</td>
<td>Raised if there was a problem in counting the entries.</td>
</tr>
</tbody>
</table>

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 20-20
- "next_entry Function" on page 20-21

first_attribute Function
This function fetches the first attribute of a given entry in the result set.

Syntax
```sql
DBMS_LDAP.first_attribute (  
    ld IN SESSION,  
    msg IN MESSAGE,  
    ber_elem OUT BER_ELEMENT)  
RETURN VARCHAR2;
```
first_attribute Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The entry whose attributes are to be stepped through, as returned by first_entry or next_entry</td>
</tr>
<tr>
<td>ber_elem (OUT)</td>
<td>A handle to a BER_ELEMENT that is used to keep track of which attribute in the entry has been read</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>The name of the attribute if it exists. NULL if no attribute exists or if an error occurred.</td>
</tr>
<tr>
<td>ber_elem</td>
<td>A handle used by DBMS_LDAP.next_attribute to iterate over all of the attributes</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming msg handle is invalid.</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_LDAP Subprograms

next_attribute Function

This function fetches the next attribute of a given entry in the result set.

Syntax

```sql
DBMS_LDAP.next_attribute (
    ld IN SESSION,
    msg IN MESSAGE,
    ber_elem IN BER_ELEMENT)
RETURN VARCHAR2;
```

See Also:
- "first_entry Function" on page 20-20
- "next_entry Function" on page 20-21
- "next_attribute Function" on page 20-27
- "get_values Function" on page 20-30
- "get_values_len Function" on page 20-32
### Parameters

**Table 20–37 next_attribute Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The entry whose attributes are to be stepped through, as returned by <code>first_entry</code> or <code>next_entry</code>.</td>
</tr>
<tr>
<td>ber_elem (IN)</td>
<td>A handle to a BER ELEMENT that is used to keep track of which attribute in the entry has been read.</td>
</tr>
</tbody>
</table>

### Return Values

**Table 20–38 next_attribute Function Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>The name of the attribute, if it exists.</td>
</tr>
</tbody>
</table>

### Exceptions

**Table 20–39 next_attribute Function Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle <code>ld</code> is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming <code>msg</code> handle is invalid.</td>
</tr>
</tbody>
</table>

### 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.
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

```sql
DBMS_LDAP.get_dn (  
    ld  IN SESSION,  
    msg  IN MESSAGE)  
RETURN VARCHAR2;
```
get_values Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>msg (IN)</td>
<td>The entry whose DN is to be returned.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>The X.500 distinguished name of the entry as a PL/SQL string. NULL if there was a problem.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming msg handle is invalid.</td>
</tr>
<tr>
<td>get_dn_error</td>
<td>Raised if there was a problem in determining the DN.</td>
</tr>
</tbody>
</table>

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 20-52.

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, 
) 
```
Summary of DBMS_LDAP Subprograms

```sql
ldapentry IN MESSAGE,
attr IN VARCHAR2)
RETURN STRING_COLLECTION;
```

## Parameters

**Table 20–43 get_values Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>ldapentry (IN)</td>
<td>A valid handle to an entry returned from a search result.</td>
</tr>
<tr>
<td>attr (IN)</td>
<td>The name of the attribute for which values are being sought.</td>
</tr>
</tbody>
</table>

## Return Values

**Table 20–44 get_values Function Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING_COLLECTION</td>
<td>A PL/SQL string collection containing all of the values of the given attribute.</td>
</tr>
<tr>
<td></td>
<td>NULL if there are no values associated with the given attribute.</td>
</tr>
</tbody>
</table>

## Exceptions

**Table 20–45 get_values Function Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid message</td>
<td>Raised if the incoming entry handle is invalid.</td>
</tr>
</tbody>
</table>

## 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`. 
get_values_len Function

See Also:
- "first_entry Function" on page 20-20
- "next_entry Function" on page 20-21
- "get_values_len Function" on page 20-32
- "count_values Function" on page 20-48

get_values_len Function
This function retrieves values of attributes that have a Binary syntax.

Syntax

```sql
DBMS_LDAP.get_values_len (  
   ld IN SESSION,  
   ldapentry IN MESSAGE,  
   attr IN VARCHAR2)  
RETURN BINVAL_COLLECTION;
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>ldapentrymsg (IN)</td>
<td>A valid handle to an entry returned from a search result.</td>
</tr>
<tr>
<td>attr (IN)</td>
<td>The string name of the attribute for which values are being sought.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINVAL_COLLECTION</td>
<td>A PL/SQL Raw collection containing all the values of the given attribute.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL if there are no values associated with the given attribute.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_message</td>
<td>Raised if the incoming entry handle is invalid.</td>
</tr>
</tbody>
</table>

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 non-binary attribute values.
delete_s Function

**See Also:**
- "first_entry Function" on page 20-20
- "next_entry Function" on page 20-21
- "get_values Function" on page 20-30
- "count_values_len Function" on page 20-49

**delete_s Function**

This function removes a leaf entry in the LDAP Directory Information Tree.

**Syntax**

```sql
DBMS_LDAP.delete_s (  
  ld IN SESSION,  
  entrydn IN VARCHAR2)  
RETURN PLS_INTEGER;
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session</td>
</tr>
<tr>
<td>entrydn (IN)</td>
<td>The X.500 distinguished name of the entry to delete.</td>
</tr>
</tbody>
</table>
Return Values

Table 20–50  delete_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP.SUCCESS if the delete operation was successful. An exception is raised otherwise.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–51  delete_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>Raised if the distinguished name of the entry is invalid.</td>
</tr>
<tr>
<td>general_error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 non-leaf entries.

See Also:  "modrdn2_s Function" on page 20-35.

modrdn2_s Function

This function modrdn2_s can be used to rename the relative distinguished name of an entry.

Syntax

```sql
DBMS_LDAP.modrdn2_s (  
    ld IN SESSION,  
    entrydn IN VARCHAR2  
    newrdn IN VARCHAR2  
    deleteoldrdn IN PLS_INTEGER)  
RETURN PLS_INTEGER;
```
modrdn2_s Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A valid LDAP session handle.</td>
</tr>
<tr>
<td>entrydn (IN)</td>
<td>The distinguished name of the entry. (This entry must be a leaf node in the DIT.).</td>
</tr>
<tr>
<td>newrdn (IN)</td>
<td>The new relative distinguished name of the entry.</td>
</tr>
<tr>
<td>deleteoldrdn(IN)</td>
<td>A boolean value that if non-zero, indicates that the attribute values from the old name should be removed from the entry.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Raised if the session handle ld is invalid.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>Raised if the distinguished name of the entry is invalid.</td>
</tr>
<tr>
<td>invalid_rdn</td>
<td>Invalid LDAP RDN.</td>
</tr>
<tr>
<td>invalid_deleteoldrdn</td>
<td>Invalid LDAP deleteoldrdn.</td>
</tr>
<tr>
<td>general error</td>
<td>For all other errors. The error string associated with this exception explains the error in detail.</td>
</tr>
</tbody>
</table>

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 20-50.

err2string Function

This function converts an LDAP error code to string in the local language in which the API is operating.

Syntax

```sql
DBMS_LDAP.err2string ( ldap_err IN PLS_INTEGER )
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldap_err</td>
<td>An error number returned from one of the API calls.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>A character string appropriately translated to the local language which describes the error in detail.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>None.</td>
</tr>
</tbody>
</table>

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 20–58 create_mod_array Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num (IN)</td>
<td>The number of the attributes that you want to add or modify.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–59 create_mod_array Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD_ARRAY</td>
<td>The data structure holds a pointer to an LDAP mod array. NULL if there was a problem.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–60 create_mod_array Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>No LDAP specific exception is raised</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_LDAP Subprograms

populate_mod_array (String Version) Procedure

This procedure populates one set of attribute information for add or modify operations.

Syntax

```sql
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);
```

See Also:

- "populate_mod_array (String Version) Procedure" on page 20-39
- "modify_s Function" on page 20-43
- "add_s Function" on page 20-45
- "free_mod_array Procedure" on page 20-47
populate_mod_array (String Version) Procedure

Parameters

Table 20–61  populate_mod_array (String Version) Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modptr (IN)</td>
<td>The data structure holds a pointer to an LDAP mod array.</td>
</tr>
<tr>
<td>Mod_op (IN)</td>
<td>This field specifies the type of modification to perform.</td>
</tr>
<tr>
<td>Mod_type (IN)</td>
<td>This field indicates the name of the attribute type to which the modification applies.</td>
</tr>
<tr>
<td>Modval (IN)</td>
<td>This field specifies the attribute values to add, delete, or replace. It is for the string values only.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–62  populate_mod_array (String Version) Procedure Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Exceptions

Table 20–63  populate_mod_array (String Version) Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_mod_array</td>
<td>Invalid LDAP mod array.</td>
</tr>
<tr>
<td>invalid_mod_option</td>
<td>Invalid LDAP mod option.</td>
</tr>
<tr>
<td>invalid_mod_type</td>
<td>Invalid LDAP mod type.</td>
</tr>
<tr>
<td>invalid_mod_value</td>
<td>Invalid LDAP mod value.</td>
</tr>
</tbody>
</table>

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.
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

```sql
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);
```
populate_mod_array (Binary Version) Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modptr (IN)</td>
<td>The data structure holds a pointer to an LDAP mod array.</td>
</tr>
<tr>
<td>Mod_op (IN)</td>
<td>This field specifies the type of modification to perform.</td>
</tr>
<tr>
<td>Mod_typ (IN)</td>
<td>This field indicates the name of the attribute type to which the modification applies.</td>
</tr>
<tr>
<td>Modval (IN)</td>
<td>This field specifies the attribute values to add, delete, or replace. It is for the binary values.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_mod_array</td>
<td>Invalid LDAP mod array.</td>
</tr>
<tr>
<td>invalid_mod_option</td>
<td>Invalid LDAP mod option.</td>
</tr>
<tr>
<td>invalid_mod_type</td>
<td>Invalid LDAP mod type.</td>
</tr>
<tr>
<td>invalid_mod_value</td>
<td>Invalid LDAP mod value.</td>
</tr>
</tbody>
</table>

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.
modify_s Function

This function performs a synchronous modification of an existing LDAP directory entry.

Syntax

```plsql
DBMS_LDAP.modify_s (  ld IN DBMS_LDAP.SESSION,  entrydn IN VARCHAR2,  modptr IN DBMS_LDAP.MOD_ARRAY)  RETURN PLS_INTEGER;
```

See Also:
- "create_mod_array Function" on page 20-38
- "modify_s Function" on page 20-43
- "add_s Function" on page 20-45
- "free_mod_array Procedure" on page 20-47
modify_s Function

Parameters

Table 20–67 modify_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.</td>
</tr>
<tr>
<td>entrydn (IN)</td>
<td>Specifies the name of the directory entry whose contents are to be modified.</td>
</tr>
<tr>
<td>modptr (IN)</td>
<td>The handle to an LDAP mod structure, as returned by a successful call to DBMS_LDAP.create_mod_array.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–68 modify_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>The indication of the success or failure of the modification operation</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–69 modify_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Invalid LDAP session.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>Invalid LDAP entry dn.</td>
</tr>
<tr>
<td>invalid_mod_array</td>
<td>Invalid LDAP mod array.</td>
</tr>
</tbody>
</table>

Usage Notes

This function call has to follow successful calls of DBMS_LDAP.create_mod_array and DBMS_LDAP.populate_mod_array.
Summary of DBMS_LDAP Subprograms

See Also:
- "create_mod_array Function" on page 20-38
- "populate_mod_array (String Version) Procedure" on page 20-39
- "add_s Function" on page 20-45
- "free_mod_array Procedure" on page 20-47

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

```sql
DBMS_LDAP.add_s (
    ld IN DBMS_LDAP.SESSION,
    entrydn IN VARCHAR2,
    modptr IN DBMS_LDAP.MOD_ARRAY)
RETURN PLS_INTEGER;
```
add_s Function

Parameters

Table 20–70  add_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.</td>
</tr>
<tr>
<td>Entrydn (IN)</td>
<td>Specifies the name of the directory entry to be created.</td>
</tr>
<tr>
<td>Modptr (IN)</td>
<td>The handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–71  add_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>The indication of the success or failure of the modification operation.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–72  add_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Invalid LDAP session.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>Invalid LDAP entry dn.</td>
</tr>
<tr>
<td>invalid_mod_array</td>
<td>Invalid LDAP mod array.</td>
</tr>
</tbody>
</table>

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.
free_mod_array Procedure

This procedure frees the memory allocated by DBMS_LDAP.create_mod_array.

Syntax

```sql
DBMS_LDAP.free_mod_array (modptr IN DBMS_LDAP.MOD_ARRAY);
```
count_values Function

Parameters

Table 20–73  free_mod_array Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modptr (in)</td>
<td>The handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–74  free_mod_array Procedure Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Exceptions

Table 20–75  free_mod_array Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>No LDAP specific exception is raised.</td>
</tr>
</tbody>
</table>

See Also:

- "create_mod_array Function" on page 20-38
- "populate_mod_array (String Version) Procedure" on page 20-39
- "modify_s Function" on page 20-43
- "add_s Function" on page 20-45

count_values Function

This function counts the number of values returned by DBMS_LDAP.get_values.

Syntax

```plsql
DBMS_LDAP.count_values (  
    values IN DBMS_LDAP.STRING_COLLECTION
) RETURN PLS_INTEGER;
```
count_values_len Function

This function counts the number of values returned by DBMS_LDAP.get_values_len.

Syntax

```plaintext
dbms_ldap.count_values_len (  
    values IN dbms_ldap.binval_collection  
) RETURN pls_integer;
```
rename_s Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values (IN)</td>
<td>The collection of binary values.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>The indication of the success or failure of the operation.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>No LDAP specific exception is raised.</td>
</tr>
</tbody>
</table>

See Also:

- "get_values_len Function" on page 20-32
- "count_values Function" on page 20-48

rename_s Function

This function renames an LDAP entry synchronously.

Syntax

```sql
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 20–82 rename_s Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.</td>
</tr>
<tr>
<td>Dn (IN)</td>
<td>Specifies the name of the directory entry to be renamed or moved.</td>
</tr>
<tr>
<td>newrdn (IN)</td>
<td>Specifies the new RDN.</td>
</tr>
<tr>
<td>Newparent (IN)</td>
<td>Specifies the DN of the new parent.</td>
</tr>
<tr>
<td>Deleteoldrdn (IN)</td>
<td>Specifies if the old RDN should be retained. If this value is 1, then the old RDN is removed.</td>
</tr>
<tr>
<td>Serverctrls (IN)</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>Clientctrls (IN)</td>
<td>Currently not supported.</td>
</tr>
</tbody>
</table>

Return Values

Table 20–83 rename_s Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>The indication of the success or failure of the operation.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–84 rename_s Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Invalid LDAP Session.</td>
</tr>
<tr>
<td>invalid_entry_dn</td>
<td>Invalid LDAP DN.</td>
</tr>
<tr>
<td>invalid_rdn</td>
<td>Invalid LDAP RDN.</td>
</tr>
<tr>
<td>invalid_newparent</td>
<td>Invalid LDAP newparent.</td>
</tr>
<tr>
<td>invalid_deleteoldrdn</td>
<td>Invalid LDAP deleteoldrdn.</td>
</tr>
</tbody>
</table>

See Also: "modrdn2_s Function" on page 20-35.
**explode_dn Function**

This function breaks a DN up into its components.

**Syntax**

```sql
DBMS_LDAP.explode_dn (  
   dn   IN VARCHAR2,  
   notypes   IN PLS_INTEGER)  
RETURN STRING_COLLECTION;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn (IN)</td>
<td>Specifies the name of the directory entry to be broken up.</td>
</tr>
<tr>
<td>Notypes (IN)</td>
<td>Specifies if the attribute tags will be returned. If this value is not 0, no attribute tags are returned.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING_COLLECTION</td>
<td>An array of strings. If the DN cannot be broken up, NULL is returned.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_entry_dn</td>
<td>Invalid LDAP DN.</td>
</tr>
<tr>
<td>invalid_notypes</td>
<td>Invalid LDAP notypes value.</td>
</tr>
</tbody>
</table>

**See Also:** ['get_dn Function' on page 20-29.]

**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,  
   sslwr1  IN VARCHAR2,  
   sslwalletpasswd  IN VARCHAR2,  
   sslauth  IN PLS_INTEGER)  
RETURN PLS_INTEGER;
open_ssl Function

Parameters

Table 20–88 open_ssl Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (IN)</td>
<td>A handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init.</td>
</tr>
<tr>
<td>Sslwr1 (IN)</td>
<td>Specifies the wallet location (Required for one-way or two-way SSL connection.)</td>
</tr>
<tr>
<td>sslwalletpasswd (IN)</td>
<td>Specifies the wallet password (Required for one-way or two-way SSL connection.)</td>
</tr>
<tr>
<td>sslauth (IN)</td>
<td>Specifies the SSL Authentication Mode (1 for no authentication required, 2 for one way authentication required, 3 for two way authentication required.)</td>
</tr>
</tbody>
</table>

Return Values

Table 20–89 open_ssl Function Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS_INTEGER</td>
<td>The indication of the success or failure of the operation.</td>
</tr>
</tbody>
</table>

Exceptions

Table 20–90 open_ssl Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_session</td>
<td>Invalid LDAP Session.</td>
</tr>
<tr>
<td>invalid_ssl_wallet_loc</td>
<td>Invalid LDAP SSL wallet location.</td>
</tr>
<tr>
<td>invalid_ssl_wallet_passwd</td>
<td>Invalid LDAP SSL wallet passwd.</td>
</tr>
<tr>
<td>invalid_ssl_auth_mode</td>
<td>Invalid LDAP SSL authentication mode.</td>
</tr>
</tbody>
</table>

Usage Notes

Call DBMS_LDAP.init first to acquire a valid LDAP session.

See Also: "init Function" on page 20-6.
The PL/SQL package 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
- Summary of DBMS_LIBCACHE Subprograms
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 through roles. This is a standard PL/SQL security limitation.

Summary of DBMS_LIBCACHE Subprograms

**Table 21–1 DBMS_SESSION Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;COMPILE_CURSORS_FROM_REMOTE Procedure&quot; on page 21-2</td>
<td>Extracts SQL in batch from the source instance and compiles the SQL at the target instance.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Link Name</td>
<td>The database link pointing to the instance used for extracting the SQL statements.</td>
</tr>
<tr>
<td>Source username</td>
<td>Parsing username for the SQL statements extracted.</td>
</tr>
<tr>
<td>Execution threshold</td>
<td>Lower bound on the number of executions. Below this value cursors will not be selected for compiling.</td>
</tr>
</tbody>
</table>
### 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 precompiling 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. The default value (1000) allows you to skip cursors that are invalid and so never executed.

---

**Table 21–2  COMPILE_CURSORS_FROM_REMOTE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharable memory</td>
<td>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.</td>
</tr>
<tr>
<td>threshold</td>
<td></td>
</tr>
</tbody>
</table>

---
COMPILE_CURSORS_FROM_REMOTE Procedure
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
- Datatypes, Constants, and Exceptions for DBMS_LOB
- Security for DBMS_LOB
- Rules and Limitations for DBMS_LOB
- Temporary LOBs
- Summary of DBMS_LOB Subprograms
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 filesystem. See also Chapter 1 of Oracle9i Application Developer’s Guide - Large Objects (LOBs).

Internal LOBs
For internal LOBs, you must first use SQL data definition language (DDL) to define tables that contain LOB columns and then use SQL data manipulation language (DML) to initialize or populate the locators in these LOB columns.

External LOBs
For external LOBs, you must ensure that a DIRECTORY object representing a valid, existing physical directory has been defined, and that physical files exist with read permission for Oracle. If your operating system uses case-sensitive pathnames, then be sure you specify the directory in the correct format.

After the LOBs are defined and created, you may then SELECT 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.

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>
<thead>
<tr>
<th>Table 22–1  DBMS_LOB datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLOB</strong></td>
</tr>
<tr>
<td><strong>RAW</strong></td>
</tr>
<tr>
<td><strong>CLOB</strong></td>
</tr>
</tbody>
</table>
The DBMS_LOB package defines no special types. NCLOB is a special case of CLOBs for fixed-width and varying-width, multi-byte 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:

```sql
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>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_argval</td>
<td>21560</td>
<td>The argument is expecting a non-NULL, valid value but the argument value passed in is NULL, invalid, or out of range.</td>
</tr>
</tbody>
</table>
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 Oracle8i, 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.

### Table 22–2  DBMS_LOB Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_error</td>
<td>22925</td>
<td>You are trying to write too much data to the LOB; LOB size is limited to 4 gigabytes.</td>
</tr>
<tr>
<td>noexist_directory</td>
<td>22285</td>
<td>The directory leading to the file does not exist.</td>
</tr>
<tr>
<td>nopriv_directory</td>
<td>22286</td>
<td>The user does not have the necessary access privileges on the directory alias and/or the file for the operation.</td>
</tr>
<tr>
<td>invalid_directory</td>
<td>22287</td>
<td>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.</td>
</tr>
<tr>
<td>operation_failed</td>
<td>22288</td>
<td>The operation attempted on the file failed.</td>
</tr>
<tr>
<td>unopened_file</td>
<td>22289</td>
<td>The file is not open for the required operation to be performed.</td>
</tr>
<tr>
<td>open_toomany</td>
<td>22290</td>
<td>The number of open files has reached the maximum limit.</td>
</tr>
</tbody>
</table>
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, non-zero values are permitted for the parameters that represent size and positional quantities, such as amount, offset, newlen, nth, etc. 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 multi-byte 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; i.e., they must both be NCLOBs, or they must both be CLOBs.

- If the value of amount plus the offset exceeds 4 GB (i.e., lobmaxsize+1) for BLOBs and BFILES, and (lobmaxsize/character_width_in_bytes)+1 for CLOBs in calls to update subprograms (i.e., 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 FILEEXIST, 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 above 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

Oracle8i 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 Oracle8i, 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 LOBs

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 per temporary LOB. The temporary LOB locator can be passed "by ref" 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
Temporary LOBs

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 per session in a v$ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist per session. V$ 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, etc.) 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 OCILobLocatatorAssign 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.

```sql
DECLARE
    a blob;
```
**Temporary LOBs**

```sql
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.

```sql
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 program block above, 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`.

```sql
BEGIN
  y clob;
END;
/
BEGIN
  dbms_lob.createtemporary(package.y, TRUE);
END;
```
Summary of DBMS_LOB Subprograms

See Also: PL/SQL User’s Guide and Reference, for more information on NOCOPY syntax

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>21560</td>
<td>Value for argument %s is not valid.</td>
</tr>
<tr>
<td>ACCESS_ERROR</td>
<td>22925</td>
<td>Attempt to read or write beyond maximum LOB size on %s.</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td></td>
<td>EndofLob indicator for looping read operations. This is not a hard error.</td>
</tr>
<tr>
<td>VALUE_ERROR</td>
<td>6502</td>
<td>PL/SQL error for invalid values to subprogram’s parameters.</td>
</tr>
</tbody>
</table>

Summary of DBMS_LOB Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;APPEND Procedure&quot; on page 22-15</td>
<td>Appends the contents of the source LOB to the destination LOB.</td>
</tr>
<tr>
<td>&quot;CLOSE Procedure&quot; on page 22-16</td>
<td>Closes a previously opened internal or external LOB.</td>
</tr>
<tr>
<td>&quot;COMPARE Function&quot; on page 22-17</td>
<td>Compares two entire LOBs or parts of two LOBs.</td>
</tr>
<tr>
<td>&quot;COPY Procedure&quot; on page 22-20</td>
<td>Copies all, or part, of the source LOB to the destination LOB.</td>
</tr>
<tr>
<td>&quot;CREATETEMPORARY Procedure&quot; on page 22-23</td>
<td>Creates a temporary BLOB or CLOB and its corresponding index in the user’s default temporary tablespace.</td>
</tr>
<tr>
<td>&quot;ERASE Procedure&quot; on page 22-24</td>
<td>Erases all or part of a LOB.</td>
</tr>
<tr>
<td>&quot;FILECLOSE Procedure&quot; on page 22-26</td>
<td>Closes the file.</td>
</tr>
<tr>
<td>&quot;FILECLOSEALL Procedure&quot; on page 22-27</td>
<td>Closes all previously opened files.</td>
</tr>
<tr>
<td>&quot;FILEEXISTS Function&quot; on page 22-28</td>
<td>Checks if the file exists on the server.</td>
</tr>
</tbody>
</table>
### Table 22–4  DBMS_LOB Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;FILEGETNAME Procedure&quot; on page 22-29</td>
<td>Gets the directory alias and file name.</td>
</tr>
<tr>
<td>&quot;FILEISOPEN Function&quot; on page 22-31</td>
<td>Checks if the file was opened using the input BFILE locators.</td>
</tr>
<tr>
<td>&quot;FILEOPEN Procedure&quot; on page 22-32</td>
<td>Opens a file.</td>
</tr>
<tr>
<td>&quot;FREETEMPORARY Procedure&quot; on page 22-33</td>
<td>Frees the temporary BLOB or CLOB in the user’s default temporary tablespace.</td>
</tr>
<tr>
<td>&quot;GETCHUNKSIZE Function&quot; on page 22-34</td>
<td>Returns the amount of space used in the LOB chunk to store the LOB value.</td>
</tr>
<tr>
<td>&quot;GETLENGTH Function&quot; on page 22-35</td>
<td>Gets the length of the LOB value.</td>
</tr>
<tr>
<td>&quot;INSTR Function&quot; on page 22-37</td>
<td>Returns the matching position of the nth occurrence of the pattern in the LOB.</td>
</tr>
<tr>
<td>&quot;ISOPEN Function&quot; on page 22-40</td>
<td>Checks to see if the LOB was already opened using the input locator.</td>
</tr>
<tr>
<td>&quot;ISTEMPORARY Function&quot; on page 22-41</td>
<td>Checks if the locator is pointing to a temporary LOB.</td>
</tr>
<tr>
<td>&quot;LOADFROMFILE Procedure&quot; on page 22-42</td>
<td>Loads BFILE data into an internal LOB.</td>
</tr>
<tr>
<td>&quot;OPEN Procedure&quot; on page 22-44</td>
<td>Opens a LOB (internal, external, or temporary) in the indicated mode.</td>
</tr>
<tr>
<td>&quot;READ Procedure&quot; on page 22-46</td>
<td>Reads data from the LOB starting at the specified offset.</td>
</tr>
<tr>
<td>&quot;SUBSTR Function&quot; on page 22-49</td>
<td>Returns part of the LOB value starting at the specified offset.</td>
</tr>
<tr>
<td>&quot;TRIM Procedure&quot; on page 22-52</td>
<td>Trims the LOB value to the specified shorter length.</td>
</tr>
<tr>
<td>&quot;WRITE Procedure&quot; on page 22-54</td>
<td>Writes data to the LOB from a specified offset.</td>
</tr>
<tr>
<td>&quot;WRITEAPPEND Procedure&quot; on page 22-56</td>
<td>Writes a buffer to the end of a LOB.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>Locator for the internal LOB to which the data is to be appended.</td>
</tr>
<tr>
<td>src_lob</td>
<td>Locator for the internal LOB from which the data is to be read.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either the source or the destination LOB is NULL.</td>
</tr>
</tbody>
</table>

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

```sql
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;
```

```sql
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 = 21;
    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.
Summary of DBMS_LOB Subprograms

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 functional 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 non-zero 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

```sql
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;
```

```sql
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;
```

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_1</td>
<td>LOB locator of first target for comparison.</td>
</tr>
<tr>
<td>lob_2</td>
<td>LOB locator of second target for comparison.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to compare.</td>
</tr>
</tbody>
</table>

Table 22-7  COMPARE Function Parameters
Returns

- INTEGER: Zero if the comparison succeeds, non-zero if not.
- NULL, if
  - amount < 1
  - amount > LOBMAXSIZE
  - offset_1 or offset_2 < 1
    * offset_1 or offset_2 > LOBMAXSIZE

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>

Examples

```sql
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;
END;
```

---

**Table 22–7**  **COMPARE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset_1</td>
<td>Offset in bytes or characters on the first LOB (origin: 1) for the comparison.</td>
</tr>
<tr>
<td>offset_2</td>
<td>Offset in bytes or characters on the first LOB (origin: 1) for the comparison.</td>
</tr>
</tbody>
</table>
```
indrical Lob to a destination internal Lob. You can specify the offsets for both the source and destinationLOBs, and the number of bytes or characters to copy.

If the offset you specify in the destinationLOB is beyond the end of the data currently in thisLOB, then zero-byte fillers or spaces are inserted in the destinationBLOB or CLOB respectively. If the offset is less than the current length of the destinationLOB, then existing data is overwritten.

It is not an error to specify an amount that exceeds the length of the data in the sourceLOB. Thus, you can specify a large amount to copy from the sourceLOB, which copies data from the src_offset to the end of the sourceLOB.
```
Summary of DBMS_LOB Subprograms

Syntax

```sql
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 22–9  COPY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the copy target.</td>
</tr>
<tr>
<td>src_lob</td>
<td>LOB locator of source for the copy.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to copy.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.</td>
</tr>
</tbody>
</table>

Exceptions

**Table 22–10  COPY Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or invalid.</td>
</tr>
</tbody>
</table>
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

```sql
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;
```

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator.</td>
</tr>
</tbody>
</table>
```
**ERASE Procedure**

This procedure erases an entire internal LOB or part of an internal LOB.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>Specifies if LOB should be read into buffer cache or not.</td>
</tr>
<tr>
<td>dur</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example**

```
DBMS_LOB.CREATETEMPORARY (Dest_Loc, TRUE)
```

**See Also:** PL/SQL User’s Guide and Reference for more information about NOCOPY and passing temporarylobs as parameters.

**Syntax**

```
DBMS_LOB.ERASE (lob_loc IN OUT NOCOPY BLOB,
                  amount IN OUT NOCOPY INTEGER,
                  offset IN INTEGER := 1);
```

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.

```
DBMS_LOB.ERASE (lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
```

**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 22-52.
Summary of DBMS_LOB Subprograms

Parameters

Table 22–12 ERASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be erased.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.</td>
</tr>
<tr>
<td>offset</td>
<td>Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–13 ERASE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any input parameter is NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1 or amount &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1 or offset &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>

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

Table 22–11 ERASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>IN OUT NOCOPY INTEGER,</td>
</tr>
<tr>
<td>offset</td>
<td>IN INTEGER := 1);</td>
</tr>
</tbody>
</table>
FILECLOSE Procedure

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 22-52

FILECLOSE Procedure

This procedure closes a BFILE that has already been opened via 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 22–14  FILECLOSE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE to be closed.</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–15  FILECLOSE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>NULL input value for file_loc.</td>
</tr>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened with the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
</tbody>
</table>
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 22-32
- "FILECLOSEALL Procedure" on page 22-27

FILECLOSEALL Procedure

This procedure closes all BFILEs opened in the session.

Syntax

DBMS_LOB.FILECLOSEALL;

Exceptions

Table 22–16 FILECLOSEALL Procedure Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>No file has been opened in the session.</td>
</tr>
</tbody>
</table>

Example

CREATE OR REPLACE PROCEDURE Example_6 IS
FILEEXISTS Function

This function finds out if a given BFILE locator points to a file that actually exists on the server’s filesystem.

Syntax

```sql
DBMS_LOB.FILEEXISTS (  
    file_loc IN BFILE)  
RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
</tbody>
</table>
Returns

| Table 22–18  FILEEXISTS Function Returns |
|-----------------|-----------------|
| Return | Description |
| 0 | Physical file does not exist. |
| 1 | Physical file exists. |

Exceptions

| Table 22–19  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

```sql
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 22-31.

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 pathname is 2000.

**Syntax**

```sql
DBMS_LOB.FILEGETNAME (  
    file_loc IN BFILE,  
    dir_alias OUT VARCHAR2,  
    filename OUT VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
<tr>
<td>dir_alias</td>
<td>Directory alias.</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the BFILE.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>dir_alias or filename are NULL.</td>
</tr>
</tbody>
</table>

**Example**

```sql
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;
```

22-30  Supplied PL/SQL Packages and Types Reference
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
</tbody>
</table>

Returns

INTEGER: 0 = file is not open, 1 = file is open

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>
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 22-28

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
<tr>
<td>open_mode</td>
<td>File access is read-only.</td>
</tr>
</tbody>
</table>
Exceptions

Table 22–25  FILEOPEN Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>file_loc or open_mode is NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>open_mode is not equal to FILE_READONLY.</td>
</tr>
<tr>
<td>OPEN_TOOMANY</td>
<td>Number of open files in the session exceeds session_max_open_files.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory associated with file_loc does not exist.</td>
</tr>
<tr>
<td>INVALIDDIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>

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 22-26
- "FILECLOSEALL Procedure" on page 22-27

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**

```sql
DBMS_LOB.FREETEMPORARY (    lob_loc IN OUT NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (    lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator.</td>
</tr>
</tbody>
</table>

**Example**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lob_loc</code></td>
<td>LOB locator.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_LOB.GETLENGTH (  
    lob_loc IN BLOB)  
RETURN INTEGER;
```

```sql
DBMS_LOB.GETLENGTH (  
    lob_loc IN CLOB CHARACTER SET ANY_CS)  
RETURN INTEGER;
```

```sql
DBMS_LOB.GETLENGTH (  
    file_loc IN BFILE)  
RETURN INTEGER;
```

**Pragmas**

```sql
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB whose length is to be returned.</td>
</tr>
</tbody>
</table>

**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 OS privileges
- lob_loc cannot be read because of an OS read error

**Examples**

```sql
CREATE OR REPLACE PROCEDURE Example_11a IS
    lobd BLOB;
    length INTEGER;
BEGIN
```

22-36   Supplied PL/SQL Packages and Types Reference
-- 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.');</p>
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 {
    lobs                                       IN   BLOB,
    pattern                                   IN   RAW,
    offset                                    IN   INTEGER := 1,
    nth                                        IN   INTEGER := 1)
```
**INSTR Function**

```sql
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**

```sql
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be examined.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
<tr>
<td>pattern</td>
<td>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.</td>
</tr>
<tr>
<td>offset</td>
<td>Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)</td>
</tr>
<tr>
<td>nth</td>
<td>Occurrence number, starting at 1.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.</td>
</tr>
</tbody>
</table>
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;

Table 22–30  INSTR Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>-any one or more of the IN parameters was NULL or INVALID.</td>
</tr>
<tr>
<td></td>
<td>-offset &lt; 1 or offset &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>-nth &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>-nth &gt; LOBMAXSIZE.</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–31  INSTR Function Exceptions for BFILES

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>

Examples
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 22-49

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

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**

```sql
DBMS_LOB.ISTEMPORTARY (  
    lob_loc IN BLOB)  
RETURN INTEGER;
```

```sql
DBMS_LOB.ISTEMPORTARY (  
    lob_loc IN CLOB CHARACTER SET ANY_CS)  
RETURN INTEGER;
```
LOADFROMFILE Procedure

Pragmas

PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator.</td>
</tr>
<tr>
<td>temporary</td>
<td>Boolean, which indicates whether the LOB is temporary or not.</td>
</tr>
</tbody>
</table>

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);  
```

```
DBMS_LOB.LOADFROMFILE (  
dest_lob  IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
src_file  IN BFILE,  
amount    IN INTEGER,  
dest_offset IN INTEGER := 1,  
src_offset IN INTEGER := 1);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the target for the load.</td>
</tr>
<tr>
<td>src_file</td>
<td>BFILE locator of the source for the load.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to load from the BFILE.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Offset in bytes in the source BFILE (origin: 1) for the start of the load.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 22–35 LOADFROMFILE Procedure Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>VALUE_ERROR</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
</tr>
<tr>
<td>- src_offset or dest_offset &lt; 1.</td>
</tr>
<tr>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td>- amount &lt; 1.</td>
</tr>
<tr>
<td>- amount &gt; LOBMAXSIZE.</td>
</tr>
</tbody>
</table>

Example

```sql
CREATE OR REPLACE PROCEDURE Example_l2f IS
  lobd      BLOB;
  files     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(files, dbms_lob.file_readonly);
  dbms_lob.loadfromfile(lobd, files, amt);
  COMMIT;
  dbms_lob.fileclose(files);
END;
```

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.
In Oracle8.0, the constant `file_readonly` was the only valid mode in which to open a `BFILE`. For Oracle 8i, two new constants have been added to the `DBMS_LOB` package: `lob_readonly` and `lob_readwrite`.

**Syntax**

```sql
DBMS_LOB.OPEN (    
  lob_loc IN OUT NOCOPY BLOB,    
  open_mode IN BINARY_INTEGER);
```

```sql
DBMS_LOB.OPEN (    
  lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,    
  open_mode IN BINARY_INTEGER);
```

```sql
DBMS_LOB.OPEN (    
  file_loc IN OUT NOCOPY BFILE,    
  open_mode IN BINARY_INTEGER := file_readonly);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator.</td>
</tr>
<tr>
<td>open_mode</td>
<td>Mode in which to open.</td>
</tr>
</tbody>
</table>

**Usage Notes**

`OPEN` requires a round-trip 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 functional 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be read.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
</tbody>
</table>
Table 22–37  READ Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).</td>
</tr>
<tr>
<td>buffer</td>
<td>Output buffer for the read operation.</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–38  READ Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of lob_loc, amount, or offset parameters are NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; MAXBUFSIZE</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- offset &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td></td>
<td>- amount is greater, in bytes or characters, than the capacity of buffer.</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 22–39  READ Procedure Exceptions for BFILEs

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File is not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>
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;           
```
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 I/O on OS 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.
SUBSTR Function

Syntax

```sql
DBMS_LOB.SUBSTR (  
    lob_loc IN BLOB,  
    amount IN INTEGER := 32767,  
    offset IN INTEGER := 1)  
RETURN RAW;
```

```sql
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;
```

```sql
DBMS_LOB.SUBSTR (  
    file_loc IN BFILE,  
    amount IN INTEGER := 32767,  
    offset IN INTEGER := 1)  
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

**Table 22–40 SUBSTR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be read.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to be read.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).</td>
</tr>
</tbody>
</table>

Returns

**Table 22–41 SUBSTR Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Function overloading that has a BLOB or BFILE in parameter.</td>
</tr>
</tbody>
</table>
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.
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 22-37
- "READ Procedure" on page 22-46

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the internal LOB whose length is to be trimmed.</td>
</tr>
<tr>
<td>newlen</td>
<td>New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>lob_loc is NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- new_len &lt; 0</td>
</tr>
<tr>
<td></td>
<td>- new_len &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>

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 22-24
- "WRITEAPPEND Procedure" on page 22-56

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 22–45  WRITE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the internal LOB to be written to.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.</td>
</tr>
<tr>
<td>buffer</td>
<td>Input buffer for the write.</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–46  WRITE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; MAXBUFSIZE</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- offset &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>

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

```sql
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 22-15
- "COPY Procedure" on page 22-20

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.
Summary of DBMS_LOB Subprograms

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 22–47 WRITEAPPEND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the internal LOB to be written to.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.</td>
</tr>
<tr>
<td>buffer</td>
<td>Input buffer for the write.</td>
</tr>
</tbody>
</table>

Exceptions

Table 22–48 WRITEAPPEND Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.</td>
</tr>
</tbody>
</table>
| 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.
WRITEAPPEND Procedure

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

```sql
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 22-15
- "COPY Procedure" on page 22-20
- "WRITE Procedure" on page 22-54
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
- Summary of DBMS_LOCK Subprograms
Requirements, Security, and Constants for DBMS_LOCK

Requirements
DBMS_LOCK is most efficient with a limit of a few hundred locks per 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 -> "NuLi", 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 acquired 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 acquired 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:
maxwait constant integer := 32767;

The constant maxwait waits forever.

Summary of DBMS_LOCK Subprograms

Table 23–1 Lock Compatibility

<table>
<thead>
<tr>
<th>HELD MODE</th>
<th>GET NL</th>
<th>GET SS</th>
<th>GET SX</th>
<th>GET S</th>
<th>GET SSX</th>
<th>GET X</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
</tr>
<tr>
<td>SS</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
</tr>
<tr>
<td>SX</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>S</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>SSX</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>X</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
</tbody>
</table>

Table 23–2 DBMS_LOCK Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ALLOCATE_UNIQUE Procedure&quot; on page 23-3</td>
<td>Allocates a unique lock ID to a named lock.</td>
</tr>
<tr>
<td>&quot;REQUEST Function&quot; on page 23-5</td>
<td>Requests a lock of a specific mode.</td>
</tr>
<tr>
<td>&quot;CONVERT Function&quot; on page 23-7</td>
<td>Converts a lock from one mode to another.</td>
</tr>
<tr>
<td>&quot;RELEASE Function&quot; on page 23-8</td>
<td>Releases a lock.</td>
</tr>
<tr>
<td>&quot;SLEEP Procedure&quot; on page 23-9</td>
<td>Puts a procedure to sleep for a specific time.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_LOCK.ALLOCATE_UNIQUE (
  lockname IN VARCHAR2,
  lockhandle OUT VARCHAR2,
  expiration_secs IN INTEGER DEFAULT 864000);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lockname</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
Summary of DBMS_LOCK Subprograms

Table 23–3  ALLOCATE_UNIQUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockhandle</td>
<td>Returns the handle to the lock ID generated by ALLOCATE_UNIQUE.</td>
</tr>
<tr>
<td></td>
<td>You can use this handle in subsequent calls to REQUEST, CONVERT, and RELEASE.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>LOCKHANDLE can be up to VARCHAR2 (128).</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>expiration_specs</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```sql
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>
<thead>
<tr>
<th>Table 23–4 REQUEST Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>id or lockhandle</td>
</tr>
<tr>
<td>lockmode</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>timeout</td>
</tr>
<tr>
<td>release_on_commit</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Table 23–5 REQUEST Function Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return Value</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id or lockhandle</td>
<td>User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.</td>
</tr>
</tbody>
</table>
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

---

**Table 23–6 CONVERT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| lockmode  | New mode that you want to assign to the given lock. The available modes and their associated integer identifiers are listed below. 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 (ULR5)  
  3 - row exclusive mode (ULRX)  
  4 - share mode (ULS)  
  5 - share row exclusive mode (ULRX)  
  6 - exclusive mode (ULX)  
Each of these lock modes is explained in Oracle8 Concepts. |
| 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). |

---

**Table 23–7 CONVERT Function Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Timeout</td>
</tr>
<tr>
<td>2</td>
<td>Deadlock</td>
</tr>
<tr>
<td>3</td>
<td>Parameter error</td>
</tr>
<tr>
<td>4</td>
<td>Don’t own lock specified by id or lockhandle</td>
</tr>
<tr>
<td>5</td>
<td>Illegal lock handle</td>
</tr>
</tbody>
</table>

---

Return Values
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id or lockhandle</td>
<td>User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>3</td>
<td>Parameter error</td>
</tr>
<tr>
<td>4</td>
<td>Do not own lock specified by id or lockhandle</td>
</tr>
<tr>
<td>5</td>
<td>Illegal lock handle</td>
</tr>
</tbody>
</table>

**SLEEP Procedure**

This procedure suspends the session for a given period of time.

**Syntax**

```
DBMS_LOCK.SLEEP (
    seconds IN NUMBER);
```
SLEEP Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>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.</td>
</tr>
</tbody>
</table>

Example

This Pro*COBOL precompiler example shows how locks can be used to ensure that there are no conflicts when multiple people need to access a single device.

Print Check

Any cashier may 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. The one printer is opened by all the cashiers to avoid the overhead of opening and closing it for every check. This means that lines of output from multiple cashiers could become interleaved if we don’t ensure exclusive access to the printer. The DBMS_LOCK package is used to ensure exclusive access.

CHECK-PRINT

Get the lock "handle" for the printer lock:

```plaintext
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):

```plaintext
EXEC SQL EXECUTE
    BEGIN DBMS_LOCK.REQUEST ( :LOCKHANDLE );
END;/END-EXEC.
```

We now have exclusive use of the printer, print the check:

```plaintext
...
```

Unlock the printer so other people can use it:

```plaintext
EXEC SQL EXECUTE
```
BEGIN DBMS_LOCK.RELEASE (:LOCKHANDLE);

END; END-EXEC.
LogMiner allows you to 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 log files 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.

The data in redo log files is especially important for recovery, because you can use it to pinpoint when a database became corrupted. You can then use this information to recover the database to the state just prior to corruption.

See Also: *Oracle9i Database Administrator’s Guide* and *Oracle9i User-Managed Backup and Recovery Guide*

This chapter discusses the following topics:

- **DBMS_LOGMNR Constants**
- **Extracting Data Values From Redo Log Files**
- **Example of Using DBMS_LOGMNR**
- **Summary of DBMS_LOGMNR Subprograms**
DBMS_LOGMNR Constants

The following sections describe the constants for the DBMS_LOGMNR package.

Constants for ADD_LOGFILE Options Flag

**NEW**
DBMS_LOGMNR.NEW purges the existing list of redo log files, if any. Places the specified redo log file in the list of log files to be analyzed.

**ADDFILE**
DBMS_LOGMNR.ADDFILE adds the specified redo log file to the list of log files to be analyzed. Any attempts to add a duplicate file raise an exception (ORA-1289).

**REMOVEFILE**
DBMS_LOGMNR.REMOVEFILE removes the redo log file from the list of log files to be analyzed. Any attempts to remove a file that has not been previously added, raise an exception (ORA-1290).

Constants for START_LOGMNR Options Flag

**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$LOGMNR_CONTENTS to skip any corruptions in the redo log file being analyzed and continue processing. This option works only when a block in the redo log file (and not the header of the redo log file) has been corrupted. Caller should check the INFO column in the V$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 log files, 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.
Extracting Data Values From Redo Log Files

LogMiner data extraction from redo log files 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 log files to a LogMiner session, how to start LogMiner, 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 24-4.

EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
    LogFileName => '/oracle/logs/log1.f',
    Options => dbms_logmnr.NEW);
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
  LogFileName => '/oracle/logs/log2.f',
  Options => dbms_logmnr.ADDFILE);

EXECUTE DBMS_LOGMNR.START_LOGMNR(
  DictFileName => '/oracle/dictionary.ora');

SELECT sql_redo
FROM V$LOGMNR_CONTENTS
EXECUTE DBMS_LOGMNR.END_LOGMNR();

Summary of DBMS_LOGMNR Subprograms

Table 24–1 describes the procedures in the DBMS_LOGMNR supplied package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_LOGFILE Procedure&quot; on page 24-4</td>
<td>Adds a file to the existing or newly created list of archive files to process.</td>
</tr>
<tr>
<td>&quot;START_LOGMNR Procedure&quot; on page 24-5</td>
<td>Initializes the LogMiner utility.</td>
</tr>
<tr>
<td>&quot;END_LOGMNR Procedure&quot; on page 24-8</td>
<td>Finishes a LogMiner session.</td>
</tr>
<tr>
<td>&quot;MINE_VALUE Function&quot; on page 24-8</td>
<td>This function may be called for any row returned from V$LOGMNR_CONTENTS to retrieve the undo or redo column value of the column specified by the column_name input parameter to this function.</td>
</tr>
<tr>
<td>&quot;COLUMN_PRESENT Function&quot; on page 24-9</td>
<td>This function may be called for any row returned from V$LOGMNR_CONTENTS to determine if undo or redo column values exist for the column specified by the column_name input parameter to this function.</td>
</tr>
</tbody>
</table>

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 log files to be analyzed. Use the ADD_LOGFILE procedure to specify the list of redo log files to analyze.
Summary of DBMS_LOGMNR Subprograms

Syntax

```sql
DBMS_LOGMNR.ADD_LOGFILE(
    LogFileName IN VARCHAR2,
    Options IN BINARY_INTEGER default ADDFILE );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFileName</td>
<td>Name of the redo log file that must be added to the list of log files to be analyzed by this session.</td>
</tr>
<tr>
<td>Options</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- Starts a new list (DBMS_LOGMNR.NEW)</td>
</tr>
<tr>
<td></td>
<td>- Adds a file to an existing list (DBMS_LOGMNR.ADDFILE), or</td>
</tr>
<tr>
<td></td>
<td>- Removes a redo log file (DBMS_LOGMNR.REMOVEFILE)</td>
</tr>
</tbody>
</table>

See "Constants for ADD_LOGFILE Options Flag" on page 24-2.

Exceptions

- ORA-1284: logfile cannot be opened. Logfile or the directory may be non-existent or inaccessible.
- ORA-1286: logfile specified is not from the database that produced other logfiles added for analysis.
- ORA-1287: logfile specified is from a different database incarnation.
- ORA-1289: attempt to add duplicate logfile.
- ORA-1290: attempt to remove unlisted logfile.
- ORA-1337: logfile specified has a different compatibility version than the rest of the logfiles added.

START_LOGMNR Procedure

This procedure starts a LogMiner session.

Note: If you want to analyze five redo log files, you must call the ADD_LOGFILE procedure five times.
START_LOGMNR Procedure

Note: This procedure fails if you did not specify a list of redo log files to be analyzed previously through the ADD_LOGFILE procedure.

Syntax

```sql
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-1988',
    DictFileName IN VARCHAR2 default '',
    Options IN BINARY_INTEGER default 0);
```

Parameters

The parameters for the DBMS_LOGMNR.START_LOGMNR procedure are listed in Table 24–3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startScn</td>
<td>Only consider redo records with SCN greater than or equal to the startScn specified. This fails if there is no redo log file with an SCN range (i.e, the LOW_SCN and NEXT_SCN associated with the log file as shown in V$LOGMNR_LOGS view) containing the startScn.</td>
</tr>
<tr>
<td>endScn</td>
<td>Only consider redo records with SCN less than or equal to the endScn specified. This fails if there is no redo log file with an SCN range (i.e, the LOW_SCN and NEXT_SCN associated with the log file as shown in V$LOGMNR_LOGS view) containing the endScn.</td>
</tr>
<tr>
<td>startTime</td>
<td>Only consider redo records with timestamp greater than or equal to the startTime specified. This fails if there is no redo log file with a time range (i.e, the LOW_TIME and HIGH_TIME associated with the log file as shown in V$LOGMNR_LOGS view) containing the startTime. This parameter is ignored if startScn is specified.</td>
</tr>
<tr>
<td>endTime</td>
<td>Only consider redo records with timestamp less than or equal to the endTime specified. This fails if there is no redo log file with a time range (i.e, the LOW_TIME and HIGH_TIME associated with the log file as shown in V$LOGMNR_LOGS view) containing the endTime. This parameter is ignored if endScn is specified.</td>
</tr>
</tbody>
</table>
After executing the `START_LOGMNR` procedure, you can make use of the following views:

- `V$LOGMNR_DICTIONARY` - contains current information about the dictionary file
- `V$LOGMNR_PARAMETERS` - contains information about the LogMiner session
- `V$LOGMNR_LOGS` - contains information about the redo log files being analyzed

### 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 options is specified
- 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.

### Table 24–3  START_LOGMNR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DictFileName</td>
<td>This flat file contains a snapshot of the database catalog. It is used to</td>
</tr>
<tr>
<td></td>
<td>reconstruct SQL_REDO and SQL_UNDO columns in V$LOGMNR_CONTENTS, as well as</td>
</tr>
<tr>
<td></td>
<td>to fully translate SEG_NAME, SEG_OWNER, SEG_TYPE_NAME and TABLE_SPACE</td>
</tr>
<tr>
<td></td>
<td>columns. The fully qualified pathname for the dictionary file must be</td>
</tr>
<tr>
<td></td>
<td>specified (This file must have been created previously through the</td>
</tr>
<tr>
<td></td>
<td>DBMS_LOGMNR_DBUILD procedure). You only need to specify this parameter if</td>
</tr>
<tr>
<td></td>
<td>neither DICT_FROM_REDO_LOGS nor DICT_FROM_ONLINE_CATALOG is specified.</td>
</tr>
</tbody>
</table>

Options

See "Constants for START_LOGMNR Options Flag" on page 24-2.
END_LOGMNR Procedure

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 logfiles being analyzed.

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

```sql
DBMS_LOGMNR.END_LOGMNR;
```

Parameters

None.

Exception

- ORA-1307: No LogMiner session is active. The END_LOGMNR procedure was called without adding any logfiles.

MINE_VALUE Function

This function returns the value contained in the first parameter corresponding to the column name indicated in the second parameter.

The syntax for this function is as follows:

Syntax

```sql
dbms_logmnr.mine_value(
    sql_redo_undo IN RAW,
    column_name IN VARCHAR2 default '') RETURN VARCHAR2;
```

Parameters

The parameters for the MINE_VALUE function are listed in Table 24-4.
**Table 24–4 MINE_VALUE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_redo_undo</td>
<td>Value of the REDO_VALUE or UNDO_VALUE columns selected from V$LOGMNR_CONTENTS. The value of this parameter directs LogMiner to return undo or redo column information. This parameter can be thought of as a self-describing record that contains values corresponding to several columns in a table.</td>
</tr>
<tr>
<td>column_name</td>
<td>Fully qualified column name (schema.table.column) which this function will return information about.</td>
</tr>
</tbody>
</table>

**Returns**

**Table 24–5 Return Values for MINE_VALUE Function**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>The column is not contained within the self-describing record or the column value is NULL.</td>
</tr>
<tr>
<td>NON-NULL</td>
<td>The column is contained within the self-describing record; the value is returned in string format.</td>
</tr>
</tbody>
</table>

**Exceptions**

No LogMiner errors are returned.

**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.

**COLUMN_PRESENT Function**

This function is meant to be used in conjunction with DBMS_LOGMNR.MINE_VALUE. The syntax for this function is as follows:
COLUMN_PRESENT Function

Syntax

```sql
dbms_logmnr.column_present(
    sql_redo_undo   IN RAW,
    column_name     IN VARCHAR2 default '') RETURN NUMBER;
```

Parameters

The parameters for the COLUMN_PRESENT function are listed in Table 24–6.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_redo.undo</td>
<td>Value of the REDO_VALUE or UNDO_VALUE columns selected from V$LOGMNR_CONTENTS. The value of this parameter directs LogMiner to look for undo or redo column information.</td>
</tr>
<tr>
<td>column_name</td>
<td>Fully qualified column name which this function will return information about.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Specified column is not present in this row of V$LOGMNR_CONTENTS.</td>
</tr>
<tr>
<td>1</td>
<td>Column is present in this row of V$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.</td>
</tr>
</tbody>
</table>

Exceptions

No LogMiner errors are returned.

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.
COLUMNS PRESENT Function

24-12  Supplied PL/SQL Packages and Types Reference
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
- Summary of `DBMS_LOGMNR_CDC_PUBLISH` Subprograms
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 25–1 describes the procedures in the DBMS_LOGMNR_CDC_PUBLISH supplied package.
Summary of DBMS_LOGMNR_CDC_PUBLISH Subprograms

Table 25–1  DBMS_LOGMNR_CDC_PUBLISH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_CHANGE_TABLE Procedure on page 25-3</td>
<td>Creates a change table in a specified schema and creates corresponding Change Data Capture metadata.</td>
</tr>
<tr>
<td>ALTER_CHANGE_TABLE Procedure on page 25-8</td>
<td>Adds or drops columns for an existing change table, or changes the properties of an existing change table.</td>
</tr>
<tr>
<td>DROP_SUBSCRIBER_VIEW Procedure on page 25-13</td>
<td>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.</td>
</tr>
<tr>
<td>DROP_SUBSCRIPTION Procedure on page 25-14</td>
<td>Allows a publisher to drop a subscription that was created with a prior call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
<tr>
<td>DROP_CHANGE_TABLE Procedure on page 25-16</td>
<td>Drops an existing change table when there is no more activity on the table.</td>
</tr>
<tr>
<td>PURGE Procedure on page 25-17</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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-separated string.

```sql
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,
)```

DBMS_LOGMNR_CDC_PUBLISH 25-3
CREATE_CHANGE_TABLE Procedure

```
object_id IN CHAR,
source_colmap IN CHAR,
target_colmap IN CHAR,
options_string IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Name of the schema that owns the change table.</td>
</tr>
<tr>
<td>change_table_name</td>
<td>Name of the change table that is being created.</td>
</tr>
<tr>
<td>change_set_name</td>
<td>Name of an existing change set with which this change table is associated. Synchronous change tables must specify SYNC_SET.</td>
</tr>
<tr>
<td>source_schema</td>
<td>The schema where the source table is located.</td>
</tr>
<tr>
<td>source_table</td>
<td>The source table from which the change records are captured.</td>
</tr>
<tr>
<td>column_type_list</td>
<td>Comma-separated list of columns and datatypes that are being tracked.</td>
</tr>
<tr>
<td>capture_values</td>
<td>Set this parameter to one of the following capture values for update operations:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>OLD</td>
</tr>
<tr>
<td></td>
<td>NEW</td>
</tr>
<tr>
<td></td>
<td>BOTH</td>
</tr>
<tr>
<td>rs_id</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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.</td>
</tr>
</tbody>
</table>

Set this parameter to Y or N, as follows:
### Summary of DBMS_LOGMNR_CDC_PUBLISH Subprograms

**DBMS_LOGMNR_CDC_PUBLISH**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>Indicates that you want to add a column to the change table that will contain the row sequence of the change.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track the <code>rs_id</code> column.</td>
</tr>
<tr>
<td>row_id</td>
<td>Adds a column to the change table that contains the row ID of the changed row in the source table, as follows.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>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.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track the <code>row_id</code> column.</td>
</tr>
<tr>
<td>user_id</td>
<td>Adds a column to the change table that contains the user name of the user who entered a DML statement, as follows.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>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.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track users.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Adds a column to the change table that contains the capture timestamp of the change record, as follows:</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>Indicates that you want to add a column to the change table that contains the capture timestamp of the change record.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track timestamps.</td>
</tr>
<tr>
<td>object_id</td>
<td>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:</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>Indicates that you want to add a column to the change table that contains the object ID of this change record.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track object IDs.</td>
</tr>
</tbody>
</table>

---

**Table 25–2  CREATE_CHANGE_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Y</td>
<td>Indicates that you want to add a column to the change table that will contain the row sequence of the change.</td>
</tr>
<tr>
<td>N</td>
<td>Indicates that you do not want to track the <code>rs_id</code> column.</td>
</tr>
</tbody>
</table>

---

**DBMS_LOGMNR_CDC_PUBLISH 25-5**
### CREATE_CHANGE_TABLE Procedure

**Table 25–2 CREATE_CHANGE_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_colmap</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>target_colmap</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>options_string</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Exceptions

**Table 25–3 CREATE_CHANGE_TABLE Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31409</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31416</td>
<td>The value specified for the source_colmap parameter is invalid. For synchronous mode, specify either Y or N.</td>
</tr>
<tr>
<td>ORA-31417</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31418</td>
<td>While creating a synchronous change table, the name of the source schema did not match any existing schema name in the database.</td>
</tr>
<tr>
<td>ORA-31419</td>
<td>When creating a synchronous change table, the underlying source table did not exist when the procedure was called.</td>
</tr>
<tr>
<td>ORA-31420</td>
<td>When creating the first change table, a purge job is submitted to the job queue. Submission of this purge job failed.</td>
</tr>
<tr>
<td>ORA-31421</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31422</td>
<td>Owner schema does not exist.</td>
</tr>
<tr>
<td>ORA-31438</td>
<td>Duplicate change table. Re-create the change table with a unique name.</td>
</tr>
<tr>
<td>ORA-31450</td>
<td>Invalid value was specified for change_table_name.</td>
</tr>
<tr>
<td>ORA-31451</td>
<td>Invalid value was specified for the capture_value. Expecting either OLD, NEW, or BOTH.</td>
</tr>
<tr>
<td>ORA-31452</td>
<td>Invalid value was specified. Expecting either Y or N.</td>
</tr>
<tr>
<td>ORA-31459</td>
<td>System triggers for DBMS_LOGMNR_CDC_PUBLISH package are not installed.</td>
</tr>
<tr>
<td>ORA-31467</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
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-separated list.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Name of the schema that owns the change table.</td>
</tr>
<tr>
<td>change_table_name</td>
<td>Name of the change table that is being altered.</td>
</tr>
<tr>
<td>operation</td>
<td>Specifies either the value DROP or ADD to indicate whether to add or drop the columns in the field column_table or column_list.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated list of column names and datatypes for each column of the source table that should be added to, or dropped from, the change table.</td>
</tr>
<tr>
<td>rs_id</td>
<td>Adds or drops the control column that tracks the row sequence (rs_id). Set this parameter to Y or N, as follows:</td>
</tr>
<tr>
<td>row_id</td>
<td>Adds or drops a row_id column, as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the row sequence (rs_id).</td>
</tr>
<tr>
<td>N</td>
<td>The rs_id control column is not changed in the change table.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row_id</th>
<th>Adds or drops a row_id column, as follows:</th>
</tr>
</thead>
</table>
ALTER_CHANGE_TABLE Procedure

Table 25–4  ALTER_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops the row_id control column for the change table.</td>
</tr>
<tr>
<td>N</td>
<td>The row_id column is not changed in the change table.</td>
</tr>
<tr>
<td>user_id</td>
<td>Adds or drops the user name control column. Specify Y or N, as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the user_name (user_id).</td>
</tr>
<tr>
<td>N</td>
<td>The user_id column is not changed in the change table.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Adds or drops the timestamp control column to the change table, as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the timestamp.</td>
</tr>
<tr>
<td>N</td>
<td>The timestamp control column is not changed in the change table.</td>
</tr>
<tr>
<td>object_id</td>
<td>Add or drops the object_id column, as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the object_id.</td>
</tr>
<tr>
<td>N</td>
<td>The object_id control column is not changed in the change table.</td>
</tr>
<tr>
<td>source_colmap</td>
<td>Adds or drops the source_colmap control column from the change table, as</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the source_columns (source_colmap).</td>
</tr>
<tr>
<td>N</td>
<td>The source_colmap column is not changed in the change table.</td>
</tr>
<tr>
<td>target_colmap</td>
<td>Adds or drops the target_colmap control column from the change table, as</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the target_columns (target_colmap).</td>
</tr>
<tr>
<td>N</td>
<td>The target_colmap column is not changed in the change table.</td>
</tr>
</tbody>
</table>
### Summary of DBMS_LOGMNR_CDC_PUBLISH Subprograms

#### Exceptions

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.

### Table 25–4 ALTER_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>Y</td>
<td>Adds or drops a column on the change table that contains the target columns (target_colmap).</td>
</tr>
<tr>
<td>N</td>
<td>The target_colmap column is not changed in the change table.</td>
</tr>
</tbody>
</table>

### Table 25–5 ALTER_CHANGE_TABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31403</td>
<td>You issued an ALTER_CHANGE_TABLE procedure with an ADD operation but a column by this name already exists in the specified table.</td>
</tr>
<tr>
<td>ORA-31409</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31417</td>
<td>A reserved column name was specified in the column list parameter. Ensure that the name specified does not conflict with a reserved column name.</td>
</tr>
<tr>
<td>ORA-31421</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31423</td>
<td>You issued the ALTER_CHANGE_TABLE with a drop operation and the specified column does not exist in the change table.</td>
</tr>
<tr>
<td>ORA-31454</td>
<td>Illegal value was specified for operation parameter; expecting ADD or DROP.</td>
</tr>
<tr>
<td>ORA-31455</td>
<td>Nothing to alter. The specified column list is NULL and all optional control columns are N.</td>
</tr>
<tr>
<td>ORA-31456</td>
<td>An internal attempt to invoke a procedure within the DBMS_CDCUTILITY package failed. Check the trace logs for more information.</td>
</tr>
<tr>
<td>ORA-31459</td>
<td>One or more required system triggers are not installed.</td>
</tr>
</tbody>
</table>
The following table describes what happens when you add a column to a change table:

<table>
<thead>
<tr>
<th>If the publisher adds . . .</th>
<th>And . . .</th>
<th>Then . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>A user column</td>
<td>A new subscription includes this column</td>
<td>The subscription window starts at the point the column was added.</td>
</tr>
<tr>
<td>A user column</td>
<td>A new subscription does not include this newly added column</td>
<td>The subscription window starts at the low-water mark for the change table thus enabling the subscriber to see the entire table.</td>
</tr>
<tr>
<td>A user column</td>
<td>Old subscriptions exist</td>
<td>The subscription window remains unchanged and the entire table can be seen.</td>
</tr>
<tr>
<td>A control column</td>
<td>There is a new subscription</td>
<td>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.</td>
</tr>
<tr>
<td>A control column</td>
<td>—</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the <code>DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE</code> procedure.</td>
</tr>
<tr>
<td>source_schema</td>
<td>Schema name where the source table resides.</td>
</tr>
<tr>
<td>source_table</td>
<td>Name of the published source table.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.</td>
</tr>
<tr>
<td>ORA-31429</td>
<td>The subscription has not been activated. Check the subscription handle and correct it, if necessary. Call the <code>DBMS_LOGMNR_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION</code> procedure for this subscription handle and then try the original command again.</td>
</tr>
</tbody>
</table>
DROP_SUBSCRIPTION Procedure

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.

Table 25–7  DROP_SUBSCRIBER_VIEW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31432</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31433</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
DROP_CHANGE_TABLE Procedure

Example

```sql
EXECUTE DBMS_LOGMNR_CDC_PUBLISH.DROP_SUBSCRIPTION ( \
    SUBSCRIPTION_HANDLE => :subhandle);
```

DROP_CHANGE_TABLE Procedure

This procedure drops an existing change table.

Syntax

```sql
DBMS_LOGMNR_CDC_PUBLISH.DROP_CHANGE_TABLE ( 
    owner IN VARCHAR2, 
    change_table_name IN VARCHAR2, 
    force_flag IN CHAR)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Name of the schema that owns the change table.</td>
</tr>
<tr>
<td>change_table_name</td>
<td>Name of the change table that is being dropped.</td>
</tr>
<tr>
<td>force_flag</td>
<td>Drops the change table, depending on whether or not there are subscriptions making references to it, as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31421</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31422</td>
<td>Owner schema does not exist.</td>
</tr>
</tbody>
</table>
Table 25–11  DROP_CHANGE_TABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31424</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31441</td>
<td>Table is not a change table. You attempted to execute the DROP_CHANGE_TABLE procedure on a table that is not a change table.</td>
</tr>
</tbody>
</table>

Example

EXECUTE DBMS_LOGMNR_CDC_PUBLISH.DROP_CHANGE_TABLE ( \
  OWNER => 'cdc1', \
  CHANGE_TABLE_NAME => 'emp_ct' \n  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
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.

This chapter discusses the following topics:

- **Subscribing to Change Data**
- **Summary of DBMS_LOGMNR_CDC_SUBSCRIBE Subprograms**

**See Also:** *Oracle9i Data Warehousing Guide* for more information about the Oracle Change Data Capture publish and subscribe model.
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 26–1 describes the procedures for the DBMS_LOGMNR_CDC_SUBSCRIBE package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GET_SUBSCRIPTION_HANDLE Procedure&quot; on page 26-5</td>
<td>Creates a subscription handle that associates the subscription with one change set.</td>
</tr>
<tr>
<td>&quot;SUBSCRIBE Procedure&quot; on page 26-6</td>
<td>Specifies the source tables and source columns for which the subscriber wants to access change data.</td>
</tr>
<tr>
<td>&quot;ACTIVATE_SUBSCRIPTION Procedure&quot; on page 26-9</td>
<td>Indicates that a subscription is ready to start accessing change data.</td>
</tr>
<tr>
<td>&quot;EXTEND_WINDOW Procedure&quot; on page 26-10</td>
<td>Sets the subscription window boundaries (low-water and high-water mark) so that new change data can be seen.</td>
</tr>
<tr>
<td>&quot;PREPARE_SUBSCRIBER_VIEW Procedure&quot; on page 26-11</td>
<td>Creates a subscriber view in the subscriber’s schema in which the subscriber can query the change data encompassed by the current subscription window.</td>
</tr>
</tbody>
</table>
Subscribers call the procedures in the order shown in Table 26–1 unless an error occurs, at which time the subscribers should exit. Figure 26–1 shows the most common steps for using the procedures in the DBMS_LOGMNR_CDC_SUBSCRIBE package.
In Figure 26–1:

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**

```sql
DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE(
    change_set IN VARCHAR2,
    description IN VARCHAR2 := NULL,
    subscription_handle OUT NUMBER)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>change_set</td>
<td>Name of an existing change set to which the application subscribes. You must set the value to <code>SYNC_SET</code>.</td>
</tr>
<tr>
<td>description</td>
<td>Describes the subscription handle and the purpose for which it is used.</td>
</tr>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle for this subscription.</td>
</tr>
</tbody>
</table>

**Exception**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31415</td>
<td>Could not find an existing change set with this name.</td>
</tr>
<tr>
<td>ORA-31457</td>
<td>The maximum number of characters permitted in the description field was exceeded.</td>
</tr>
</tbody>
</table>
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

```sql
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.

```sql
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.

```sql
DBMS_LOGMNR_CDC_SUBSCRIBE.SUBSCRIBE (subscription_handle IN NUMBER,
publishation_id IN NUMBER,
column_list IN VARCHAR2)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
<tr>
<td>source_schema</td>
<td>Schema name where the source table resides.</td>
</tr>
<tr>
<td>source_table</td>
<td>Name of a published source table.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated list of columns from the published source table.</td>
</tr>
<tr>
<td>publication_id</td>
<td>A valid publication_id, which you can obtain from the ALL_PUBLISHED_COLUMNS view.</td>
</tr>
</tbody>
</table>

### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>The specified subscription handle does not exist, or it does not belong to this user or application.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION (
  subscription_handle IN NUMBER)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>The specified subscription handle does not exist, or it does not belong to this user ID or application.</td>
</tr>
<tr>
<td>ORA-31439</td>
<td>The subscription is already active. You can activate a subscription only once.</td>
</tr>
</tbody>
</table>

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.
EXTEND_WINDOW Procedure

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>The specified subscription handle does not exist or it does not belong to this user or application.</td>
</tr>
<tr>
<td>ORA-31429</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31430</td>
<td>The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW procedure and then try the original command again.</td>
</tr>
</tbody>
</table>

Usage Notes

- Until you call the EXTEND_WINDOW procedure to begin capturing change data, the subscription window remains empty.
Summary of DBMS_LOGMNR_CDC_SUBSCRIBE Subprograms

- 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

```sql
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the <code>GET_SUBSCRIPTION_HANDLE</code> procedure.</td>
</tr>
<tr>
<td>source_schema</td>
<td>Schema name where the source table resides.</td>
</tr>
<tr>
<td>source_table</td>
<td>Name of the published source table that belongs to the subscription handle.</td>
</tr>
<tr>
<td>view_name</td>
<td>Name of the newly-created view that will return the change data for the source table.</td>
</tr>
</tbody>
</table>
PREPARE_SUBSCRIBER_VIEW Procedure

Exceptions

Table 26–11  PREPARE_SUBSCRIBER_VIEW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>The specified subscription handle does not exist, or it does not belong to this user or application.</td>
</tr>
<tr>
<td>ORA-31429</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31430</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31432</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a</td>
</tr>
<tr>
<td></td>
<td>previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
<tr>
<td>source_schema</td>
<td>Schema name where the source table resides.</td>
</tr>
<tr>
<td>source_table</td>
<td>Name of the published source table that belongs to the subscription</td>
</tr>
<tr>
<td></td>
<td>handle.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>Subscription handle does not exist or handle does not belong to this user.</td>
</tr>
<tr>
<td></td>
<td>Call the function again with a valid subscription handle.</td>
</tr>
<tr>
<td>ORA-31429</td>
<td>The subscription has not been activated. Check the subscription handle</td>
</tr>
<tr>
<td></td>
<td>and correct it, if necessary. Call the ACTIVATE_SUBSCRIPTION procedure</td>
</tr>
<tr>
<td></td>
<td>for this subscription handle and then try the original command again.</td>
</tr>
</tbody>
</table>
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

```sql
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**

```sql
DBMS_CDC_SUBSCRIBE.PURGE_WINDOW(
  subscription_handle IN NUMBER)
```

---

**Table 26–13 DROP_SUBSCRIBER_VIEW Procedure Exceptions (Cont.)**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31432</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31433</td>
<td>The subscriber view does not exist. Either you specified an incorrect source table or its view is already dropped.</td>
</tr>
</tbody>
</table>
Parameters

Table 26–14  PURGE_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
</tbody>
</table>

Exceptions

Table 26–15  PURGE_WINDOW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.</td>
</tr>
<tr>
<td>ORA-31429</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-31430</td>
<td>The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW Procedure and then try the original command again.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription_handle</td>
<td>Unique number of the subscription handle that was returned by a previous call to the GET_SUBSCRIPTION_HANDLE procedure.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31425</td>
<td>Subscription handle does not exist or handle does not belong to this user. Call the function again with a valid subscription handle.</td>
</tr>
<tr>
<td>ORA-31430</td>
<td>The subscriber view was not dropped prior to making this call. Call the DROP_SUBSCRIBER_VIEW procedure and then try the original command again.</td>
</tr>
</tbody>
</table>

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);
```
DBMS_LOGMNR_D contains the LogMiner procedure, DBMS_LOGMNR_D.BUILD, used to create the LogMiner dictionary file. This procedure extracts the dictionary either to the redo log files or to a flat file.

This information is saved in preparation for future analysis of redo log files using the LogMiner tool.

See Also: Oracle9i Database Administrator’s Guide and Oracle9i User-Managed Backup and Recovery Guide

This chapter discusses the following topics:

- Extracting a Dictionary to the Redo Log Files
- Extracting a Dictionary to a Flat File
- Examples of Using DBMS_LOGMNR_D.BUILD
- Summary of DBMS_LOGMNR_D Subprograms
Extracting a Dictionary to the Redo Log Files

To extract a dictionary file to the redo log files, the following conditions must be met:

- The `DBMS_LOGMNR_D.BUILD` procedure must be run on a system that is running Oracle9i or later.
- Archiving mode must be enabled in order to generate usable redo.
- Oracle9i compatibility must be employed.
- The mining system must be Oracle9i or later.
- The dictionary redo files must be created from the same database that generated the redo log files you want to analyze.

The `DBMS_LOGMNR_D.BUILD` procedure will not run if there are any ongoing DDL operations.

Additionally, while the procedure is executing, no DDL operations are allowed.

Extracting a Dictionary to a Flat File

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 log files 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.
Examples of Using DBMS_LOGMNR_D.BUILD

The DBMS_LOGMNR_D package contains one procedure, DBMS_LOGMNR_D.BUILD. For a complete description of this procedure, see DBMS_LOGMNR_D.BUILD Procedure on page 27-3.

To use the DBMS_LOGMNR_D.BUILD procedure, mount and open the database whose files you will want to analyze.

Then run the PL/SQL procedure DBMS_LOGMNR_D.BUILD, as illustrated in the following examples.

Example of Extracting 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
SQLPLUS>EXECUTE dbms_logmnr_d.build('dictionary.ora',
SQLPLUS>'/oracle/database/',
SQLPLUS>options => dbms_logmnr_d.store_in_flat_file);
```

Example of Extracting to Redo Logs
```sql
SQLPLUS>EXECUTE dbms_logmnr_d.build (options => dbms_logmnr_d.store_in_redo_logs);```

Summary of DBMS_LOGMNR_D Subprograms
DBMS_LOGMNR_D contains one procedure, BUILD, which writes the dictionary tables of the current database (the online catalog) into the redo log files or into a flat file.

DBMS_LOGMNR_D.BUILD Procedure
The syntax for the DBMS_LOGMNR_D.BUILD procedure is as follows:

```
DBMS_LOGMNR_D.BUILD (
    dictionary_filename IN VARCHAR2,
    dictionary_location IN VARCHAR2,
    options IN NUMBER);
```
Parameters

Table 27–1  BUILD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary_filename</td>
<td>Name of the dictionary file</td>
</tr>
<tr>
<td>dictionary_location</td>
<td>Path to file directory</td>
</tr>
<tr>
<td>options</td>
<td>Specifies that the dictionary is written to either a flat file (STORE_IN_FLAT_FILE) or the redo log files (STORE_IN_REDO_LOGS) destination</td>
</tr>
</tbody>
</table>

To extract the dictionary to a flat file, you must supply a file name and location.

To extract the dictionary to the redo log files, 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 file name and location, without any options, the dictionary is extracted to a flat file with that name.
- If you specify a file name 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 file name 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 file name 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 log files that are to be analyzed. As of LogMiner version 9i, you can dump the dictionary to the redo log files, perform DDL operations, and dynamically apply the changes to the LogMiner dictionary.

- To monitor progress of the dictionary build issue the `SET SERVEROUTPUT ON` command.
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
- 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 XSL-T 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 browsing: GET_XML and GET_DDL return metadata for a single object and are used in SQL queries and for browsing.

This chapter discusses the following topics:

- Summary of DBMS_METADATA Subprograms
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

```
FUNCTION open
  ( object_type  IN VARCHAR2,
    version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model        IN VARCHAR2 DEFAULT 'ORACLE',
  ) RETURN NUMBER;
```
### Parameters

#### Table 28–2  Open() Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be retrieved. Table 28–3 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 Oracle9i. Future models may support a different set of object types. Most objects have names, belong to schemas, and are uniquely identified within their namespace by their schema and name. Some objects (for example, outlines) are not schema objects; these are marked with an &quot;N&quot; in Table 28–3. Some objects (for example, system privilege grants) do not have names; see the &quot;Notes&quot; column in Table 28–3. These differences are relevant when choosing object selection criteria. See &quot;SET_FILTER Procedure&quot; on page 28-5 for more information.</td>
</tr>
<tr>
<td>version</td>
<td>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 9.0.0 or higher. LATEST—the version of the metadata corresponds to the database version. A specific database version, for example, 9.0.0.</td>
</tr>
<tr>
<td>model</td>
<td>Specifies which view to use, since the API can support multiple views on the metadata. Only the ORACLE model is supported in Oracle9i.</td>
</tr>
</tbody>
</table>

#### Table 28–3  DBMS_METADATA: Object Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Schema Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION</td>
<td>stored functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX</td>
<td>indexes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEXTYPE</td>
<td>indextypes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT_GRANT</td>
<td>object grants</td>
<td></td>
<td>Not a named object.</td>
</tr>
</tbody>
</table>
OPEN Procedure

Table 28–3  DBMS_METADATA: Object Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Schema Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATOR</td>
<td>operators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTLINE</td>
<td>stored outlines</td>
<td>N</td>
<td>By default, both package specification and package body are retrieved. See &quot;SET_FILTER Procedure&quot; on page 28-5.</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>stored packages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>stored procedures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNONYM</td>
<td>synonyms</td>
<td>See notes.</td>
<td>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.</td>
</tr>
<tr>
<td>SYSTEM_GRANT</td>
<td>system privilege grants</td>
<td>N</td>
<td>Not a named object.</td>
</tr>
<tr>
<td>TABLE</td>
<td>tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIGGER</td>
<td>triggers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>user-defined types</td>
<td></td>
<td>By default, both type and type body are retrieved. See 'SET_FILTER Procedure' on page 28-5.</td>
</tr>
<tr>
<td>VIEW</td>
<td>views</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

28-4  Supplied PL/SQL Packages and Types Reference
- 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

PROCEDURE set_filter
   (handle IN NUMBER,
   name IN VARCHAR2,
   value IN VARCHAR2);
PROCEDURE set_filter
   (handle IN NUMBER,
   name IN VARCHAR2,
   value IN BOOLEAN DEFAULT TRUE);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the filter. For each filter, Table 28–5 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).</td>
</tr>
<tr>
<td>value</td>
<td>The value of the filter.</td>
</tr>
</tbody>
</table>
### Table 28–5  SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named objects</td>
<td>NAME</td>
<td>text</td>
<td>Objects with this exact name are selected.</td>
</tr>
</tbody>
</table>
|             | NAME_EXPR  | text     | The filter value is the right-hand side of a SQL comparison, i.e., a SQL comparison operator (=, !=, etc.) and the value compared against. The value must contain parentheses and quotation marks where appropriate. In particular, 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 example above, 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 (see below), then objects in that schema are selected;
    - otherwise, objects in the current schema are selected.  

| 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 28–5  **SET_FILTER: Filters**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>TABLESPACE</td>
<td>text</td>
<td>Tables in this tablespace (or having this as their default tablespace) are selected.</td>
</tr>
<tr>
<td></td>
<td>TABLESPACE_</td>
<td>text</td>
<td>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 tablespace or default tablespace 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.</td>
</tr>
<tr>
<td></td>
<td>NAME_EXPR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX, OBJECT_</td>
<td>BASE_OBJECT_</td>
<td>text</td>
<td>Indexes, triggers, or privileges are selected that are defined or granted on objects with this name. Specify SCHEMA for triggers on schemas. Specify DATABASE for database triggers.</td>
</tr>
<tr>
<td>GRANT, TRIGGER</td>
<td>NAME_SCHEMA</td>
<td>text</td>
<td>Indexes, triggers, or privileges 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.</td>
</tr>
<tr>
<td>INDEX, TRIGGER</td>
<td>SYSTEM_</td>
<td>Boolean</td>
<td>If TRUE, select indexes or triggers even if they are system-generated. If FALSE, omit system-generated indexes or triggers. Defaults to TRUE.</td>
</tr>
<tr>
<td>GRANT, SYSTEM_GRANT</td>
<td>GRANTEE</td>
<td>text</td>
<td>Privileges are selected that are granted to this user or role. Specify PUBLIC for grants to PUBLIC.</td>
</tr>
<tr>
<td>OBJECT_GRANT</td>
<td>GRANTOR</td>
<td>text</td>
<td>Privileges are selected that are granted by this user.</td>
</tr>
</tbody>
</table>

---

Summary of DBMS_METADATA Subprograms
**SET_FILTER Procedure**

---

### Table 28–5  SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNONYM</td>
<td>LONGNAME</td>
<td>text</td>
<td>A synonym name longer than 30 characters. Synonyms with this exact name are selected. If the synonym name is 30 characters or less, the NAME filter must be used.</td>
</tr>
<tr>
<td></td>
<td>LONGNAME_EXPR</td>
<td>text</td>
<td>The filter value is the right-hand side of a SQL comparison. The filter value is combined with the object attribute corresponding to the long name of the object 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 a synonym.</td>
</tr>
<tr>
<td></td>
<td>CUSTOM_FILTER</td>
<td>text</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**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

---

28-8  Supplied PL/SQL Packages and Types Reference
Summary of DBMS_METADATA Subprograms

- 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

These rules apply to dependent objects such as triggers, grants, and indexes.

- 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. SET_COUNT allows you to 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.
GET_QUERY Procedure

Syntax

```sql
PROCEDURE set_count
    ( handle IN NUMBER,
      value IN NUMBER);
```

Parameters

**Table 28–6 SET_COUNT Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>value</td>
<td>The number of objects to retrieve.</td>
</tr>
</tbody>
</table>

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

```sql
FUNCTION get_query
    (handle IN NUMBER)
    RETURN VARCHAR2;
```

Parameters

**Table 28–7 GET_QUERY Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
</tbody>
</table>
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

PROCEDURE set_parse_item
  (handle IN NUMBER,
   name IN VARCHAR2);

Parameters

Table 28–8  SET_PARSE_ITEM Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the object attribute to be parsed and returned. See Table 28–9 for the attribute object type, name, and meaning.</td>
</tr>
</tbody>
</table>
SET_PARSE_ITEM Procedure

Table 28–9  SET_PARSE_ITEM: Parse Items

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>All objects</td>
<td>VERB</td>
<td>For every row in the sys.ku$_ddls nested table returned by fetch_ddl, the verb in the corresponding ddlText is returned. See the example using sys.ku$_ddls on page 28-18.</td>
</tr>
<tr>
<td></td>
<td>OBJECT_TYPE</td>
<td>The object type as used in a DDL CREATE statement is returned, for example, TABLE or PACKAGE BODY.</td>
</tr>
<tr>
<td></td>
<td>SCHEMA</td>
<td>The object schema is returned. If the object is not a schema object, NULL is returned.</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>The object name is returned. If the object is not a named object, NULL is returned.</td>
</tr>
<tr>
<td>TABLE, INDEX</td>
<td>TABLESPACE</td>
<td>The tablespace name of the table or index is returned.</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>ENABLE</td>
<td>If the trigger is enabled, ENABLE is returned. If the trigger is disabled, DISABLE is returned.</td>
</tr>
</tbody>
</table>

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$_parsed_items nested table. See the example using sys.ku$_parsed_items on page 28-18.
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

```sql
FUNCTION add_transform
  (handle IN NUMBER,
   name IN VARCHAR2,
   encoding IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the transform. If the name is DDL, creation DDL will be generated using XSL-T 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 XSL-T stylesheet (see Oracle9i Application Developer’s Guide - XML).</td>
</tr>
<tr>
<td>encoding</td>
<td>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 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/ (see [XPATH-SUPP]), then the database character set is assumed to be the encoding.</td>
</tr>
</tbody>
</table>

See Also:

- "FETCH_xxx Procedure" on page 28-18
- Oracle9i Application Developer’s Guide - XML
ADD_TRANSFORM Procedure

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 XSL-T 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.
SET_TRANSFORM_PARAM Procedure

SET_TRANSFORM_PARAM specifies parameters to the XSL-T stylesheet identified by transform_handle. Use it to modify or customize the output of the transform.

Syntax

```
PROCEDURE set_transform_param
    (transform_handle IN NUMBER,
     name IN VARCHAR2,
     value IN VARCHAR2);

PROCEDURE set_transform_param
    (transform_handle IN NUMBER,
     name IN VARCHAR2,value IN BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transform_handle</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the parameter. Table 28–12 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).</td>
</tr>
<tr>
<td>value</td>
<td>The value of the transform.</td>
</tr>
<tr>
<td>Object Type</td>
<td>Name</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>All objects</td>
<td>PRETTY</td>
</tr>
<tr>
<td>SQLTERMINATOR</td>
<td>Boolean</td>
</tr>
<tr>
<td>TABLE</td>
<td>SEGMENT_ATTRIBUTES</td>
</tr>
<tr>
<td>TABLE</td>
<td>STORAGE</td>
</tr>
<tr>
<td>TABLE</td>
<td>TABLESPACE</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS</td>
</tr>
<tr>
<td>TABLE</td>
<td>REF_CONSTRAINTS</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS_AS_ALTER</td>
</tr>
<tr>
<td>TABLE</td>
<td>OID</td>
</tr>
<tr>
<td>TABLE</td>
<td>SIZE_BYTE_KEYWORD</td>
</tr>
<tr>
<td>INDEX</td>
<td>SEGMENT_ATTRIBUTES</td>
</tr>
<tr>
<td>INDEX</td>
<td>STORAGE</td>
</tr>
<tr>
<td>INDEX</td>
<td>TABLESPACE</td>
</tr>
</tbody>
</table>
Table 28–12  SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>SPECIFICATION</td>
<td>Boolean</td>
<td>If TRUE, emit the type specification. Defaults to TRUE.</td>
</tr>
<tr>
<td></td>
<td>BODY</td>
<td>Boolean</td>
<td>If TRUE, emit the type body. Defaults to TRUE.</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>SPECIFICATION</td>
<td>Boolean</td>
<td>If TRUE, emit the package specification. Defaults to TRUE.</td>
</tr>
<tr>
<td></td>
<td>BODY</td>
<td>Boolean</td>
<td>If TRUE, emit the package body. Defaults to TRUE.</td>
</tr>
<tr>
<td>VIEW</td>
<td>FORCE</td>
<td>Boolean</td>
<td>If TRUE, use the FORCE keyword in the CREATE VIEW statement. Defaults to TRUE.</td>
</tr>
<tr>
<td>All objects</td>
<td>DEFAULT</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>INHERIT</td>
<td>Boolean</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

XSL-T 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 28-19 for the variants.

**Syntax**

The FETCH functions and procedures are:

FUNCTION fetch_xml
  (handle IN NUMBER)
  RETURN sys.XMLType;

FUNCTION fetch_ddl
  (handle IN NUMBER)
  RETURN sys.ku$_ddls;

**See Also:** Oracle9i Application Developer’s Guide - XML, Chapter 9, “XMLType: Native Storage of XML in the Database” for a description of XMLType.

FUNCTION 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),  
  parent NUMBER );

TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item;

TYPE sys.ku$_ddl AS OBJECT (  
  item VARCHAR2(30),  
  value VARCHAR2(4000),  
  parent NUMBER );

TYPE sys.ku$_ddls IS TABLE OF sys.ku$_parsed_items;

---

28-18  Supplied PL/SQL Packages and Types Reference
DDLText CLOB,
parsedItems sys.ku$_parsed_items);
TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl;

FUNCTION fetch_clob (handle IN NUMBER)
    RETURN CLOB;
PROCEDURE fetch_clob (handle IN NUMBER,
    doc IN OUT NOCOPY CLOB);

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>doc</td>
<td>The metadata for the objects or NULL if all objects have been returned.</td>
</tr>
</tbody>
</table>

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_ARGV. 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_xxx Procedure

- `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.
- `parent`—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

PROCEDURE close (handle IN NUMBER);

Parameters

Table 28–14 CLOSE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
</tbody>
</table>

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
```sql
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 since 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, etc.
  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;  
    LOOP

-- 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,  
-- e.g., 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,
-- since 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

FUNCTION 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;

FUNCTION 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 28–15  GET_xxx Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter. In addition the following types may be specified:</td>
</tr>
<tr>
<td></td>
<td>• PACKAGE_SPEC - package specification (without body)</td>
</tr>
<tr>
<td></td>
<td>• PACKAGE_BODY - package body</td>
</tr>
<tr>
<td></td>
<td>• TYPE_SPEC - type specification (without body)</td>
</tr>
<tr>
<td></td>
<td>• TYPE_BODY - type body</td>
</tr>
<tr>
<td>name</td>
<td>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 28–5.</td>
</tr>
<tr>
<td>schema</td>
<td>A schema name (case sensitive). The default is the current schema if object_type refers to a schema object; otherwise the default is NULL.</td>
</tr>
<tr>
<td>version</td>
<td>The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.</td>
</tr>
<tr>
<td>model</td>
<td>The object model to use. This parameter takes the same values as the OPEN model parameter.</td>
</tr>
<tr>
<td>transform</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
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.

```sql
set pagesize 0
set long 90000
```
GET_XML and GET_DDL Functions

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');
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BEGIN_TABLE_REORGANIZATION Procedure</code> on page 29-4</td>
<td>Performs a process to preserve materialized view data needed for refresh.</td>
</tr>
<tr>
<td><code>END_TABLE_REORGANIZATION Procedure</code> on page 29-5</td>
<td>Ensures that the materialized view data for the master table is valid and that the master table is in the proper state.</td>
</tr>
<tr>
<td><code>EXPLAIN_MVIEW Procedure</code> on page 29-6</td>
<td>Explains what is possible with a materialized view or potential materialized view.</td>
</tr>
<tr>
<td><code>EXPLAIN_REWRITE Procedure</code> on page 29-7</td>
<td>Explains why a query failed to rewrite.</td>
</tr>
<tr>
<td><code>I_AM_A_REFRESH Function</code> on page 29-8</td>
<td>Returns the value of the <code>I_AM_REFRESH</code> package state.</td>
</tr>
<tr>
<td><code>PMARKER Function</code> on page 29-9</td>
<td>Returns a partition marker from a rowid. This function is used for Partition Change Tracking (PCT).</td>
</tr>
<tr>
<td><code>PURGE_DIRECT_LOAD_LOG Procedure</code> on page 29-9</td>
<td>Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing).</td>
</tr>
<tr>
<td><code>PURGE_LOG Procedure</code> on page 29-10</td>
<td>Purges rows from the materialized view log.</td>
</tr>
<tr>
<td><code>PURGE_MVIEW_FROM_LOG Procedure</code> on page 29-11</td>
<td>Purges rows from the materialized view log.</td>
</tr>
<tr>
<td><code>REFRESH Procedure</code> on page 29-13</td>
<td>Consistently refreshes one or more materialized views that are not members of the same refresh group.</td>
</tr>
<tr>
<td><code>REFRESH_ALL_MVIEWS Procedure</code> on page 29-16</td>
<td>Refreshes all materialized views that do not reflect changes to their master table or master materialized view.</td>
</tr>
<tr>
<td><code>REFRESH_DEPENDENT Procedure</code> on page 29-17</td>
<td>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.</td>
</tr>
<tr>
<td><code>REGISTER_MVIEW Procedure</code> on page 29-19</td>
<td>Enables the administration of individual materialized views.</td>
</tr>
</tbody>
</table>
Table 29–1   DBMS_MVIEW Package Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;UNREGISTER_MVIEW Procedure&quot; on page 29-22</td>
<td>Enables the administration of individual materialized views. Invoked at a master site or master materialized view site to unregister a materialized view.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabowner</td>
<td>Owner of the table being reorganized.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table being reorganized.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_MVIEW.END_TABLE_REORGANIZATION (  
   tabowner IN VARCHAR2,  
   tabname IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabowner</td>
<td>Owner of the table being reorganized.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table being reorganized.</td>
</tr>
</tbody>
</table>
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`:

```sql
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`:

```sql
DBMS_MVIEW.EXPLAIN_MVIEW (  
    mv IN VARCHAR2,  
    msg_array OUT SYS.ExplainMVArrayType);
```
Parameters

Table 29-4  EXPLAIN_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>The name of an existing materialized view (optionally qualified with the owner name separated by a &quot;.&quot;) or a SELECT statement for a potential materialized view.</td>
</tr>
<tr>
<td>statement_id</td>
<td>A client-supplied unique identifier to associate output rows with specific invocations of EXPLAIN_MVIEW.</td>
</tr>
<tr>
<td>msg_array</td>
<td>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.</td>
</tr>
</tbody>
</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:

```sql
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:

```sql
DBMS_MVIEW.EXPLAIN_REWRITE (
    query IN VARCHAR2(2000),
    mv IN VARCHAR2(30),
)
I_AM_A_REFRESH Function

msg_array IN OUT SYS.RewriteArrayType);

Parameters

Table 29–5 EXPLAIN_REWRITE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>SQL select statement to be explained.</td>
</tr>
<tr>
<td>mv</td>
<td>The fully qualified name of an existing materialized view in the form of SCHEMA.MV</td>
</tr>
<tr>
<td>statement_id</td>
<td>A client-supplied unique identifier to distinguish output messages</td>
</tr>
<tr>
<td>msg_array</td>
<td>The PL/SQL varray that receives the output. Use this parameter to direct EXPLAIN_REWRITE’s output to a PL/SQL VARRAY</td>
</tr>
</tbody>
</table>

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;

Parameters

None.
PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

Syntax

```sql
DBMS_MVIEW.PMARKER(rid IN ROWID)
    RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rid</td>
<td>The rowid of a row entry in a master table.</td>
</tr>
</tbody>
</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

```sql
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG();
```

Parameters

None.
PURGE_LOG Procedure

This procedure purges rows from the materialized view log.

Syntax

```sql
DBMS_MVIEW.PURGE_LOG (  
    master   IN  VARCHAR2,  
    num      IN  BINARY_INTEGER := 1,  
    flag     IN  VARCHAR2 := 'NOP');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Name of the master table or master materialized view.</td>
</tr>
<tr>
<td>num</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_MVIEW.PURGE_LOG('master_table', 2);</code></td>
</tr>
<tr>
<td></td>
<td>To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example:</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_MVIEW.PURGE_LOG('master_table',9999);</code></td>
</tr>
<tr>
<td></td>
<td>This statement completely purges the materialized view log that corresponds to <code>master_table</code> if fewer than 9999 materialized views are based on <code>master_table</code>. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.</td>
</tr>
<tr>
<td>flag</td>
<td>Specify <code>delete</code> 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 <code>num</code>. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view:</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_MVIEW.PURGE_LOG('master_table',1,'delete');</code></td>
</tr>
</tbody>
</table>
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 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 29–8  PURGE_MVIEW_FROM_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mview_id</td>
<td>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).</td>
</tr>
<tr>
<td>mviewowner</td>
<td>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.</td>
</tr>
<tr>
<td>mviewname</td>
<td>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.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>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.</td>
</tr>
</tbody>
</table>
REFRESH Procedure

This procedure refreshes a list of materialized views.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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:

```sql
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. |
<p>| 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. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge_option</td>
<td>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 <strong>PUSH</strong> with this parameter set to 2 to reduce the queue.</td>
</tr>
</tbody>
</table>
| 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_of_failures</td>
<td>Returns the number of failures that occurred during processing.</td>
</tr>
<tr>
<td>method</td>
<td>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.</td>
</tr>
<tr>
<td>rollback_seg</td>
<td>Name of the materialized view site rollback segment to use while refreshing materialized views.</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
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 29–11  REFRESH_DEPENDENT Procedure Parameters  (Page 1 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_of_failures</td>
<td>Returns the number of failures that occurred during processing.</td>
</tr>
<tr>
<td>list</td>
<td>tab</td>
</tr>
</tbody>
</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:  

```sql
DBMS_MVIEW.REFRESH_DEPENDENT
('employees,departments,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.  |
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

```sql
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); 
```

Table 29–11  REFRESH_DEPENDENT Procedure Parameters  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_refresh</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mviewowner</td>
<td>Owner of the materialized view.</td>
</tr>
<tr>
<td>mviewname</td>
<td>Name of the materialized view.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>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.</td>
</tr>
<tr>
<td>mview_id</td>
<td>The identification number of the materialized view. Specify an Oracle8 and higher materialized view as a <strong>BINARY_INTEGER</strong>. Specify an Oracle7 materialized view registering at an Oracle8 and higher master sites or master materialized view sites as a <strong>DATE</strong>.</td>
</tr>
<tr>
<td>flag</td>
<td>A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following:</td>
</tr>
<tr>
<td></td>
<td>- <code>dbms_mview.reg_rowid_mview</code> for a rowid materialized view</td>
</tr>
<tr>
<td></td>
<td>- <code>dbms_mview.reg_primary_key_mview</code> for a primary key materialized view</td>
</tr>
<tr>
<td></td>
<td>- <code>dbms_mview.reg_object_id_mview</code> for an object id materialized view</td>
</tr>
<tr>
<td></td>
<td>- <code>dbms_mview.reg_fast_refreshable_mview</code> for a materialized view that can be fast refreshed</td>
</tr>
<tr>
<td></td>
<td>- <code>dbms_mview.reg_updatable_mview</code> for a materialized view that is updatable</td>
</tr>
<tr>
<td>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:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>dbms_mview.reg_primary_key_mview + dbms_mview.reg_fast_refreshable_mview</code></td>
</tr>
<tr>
<td>You can determine the properties of a materialized view by querying the <code>ALL_MVIEWS</code> data dictionary view.</td>
<td></td>
</tr>
<tr>
<td>qry_txt</td>
<td>The first 32,000 bytes of the materialized view definition query.</td>
</tr>
</tbody>
</table>
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.

Table 29–12 REGISTER_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep_type</td>
<td>Version of the materialized view. Valid constants that can be assigned include the following:</td>
</tr>
<tr>
<td></td>
<td>- dbms_mview.reg_v7_snapshot if the materialized view is at an Oracle7 site</td>
</tr>
<tr>
<td></td>
<td>- dbms_mview.reg_v8_snapshot if the materialized view is at an Oracle8 or higher site</td>
</tr>
<tr>
<td></td>
<td>- 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</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_MVIEW.UNREGISTER_MVIEW (  
  mviewowner  IN VARCHAR2,  
  mviewname   IN VARCHAR2,  
  mviewsite   IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mviewowner</td>
<td>Owner of the materialized view.</td>
</tr>
<tr>
<td>mviewname</td>
<td>Name of the materialized view.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>Name of the materialized view site.</td>
</tr>
</tbody>
</table>
The `DBMS_OBFUSCATION_TOOLKIT` package 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 twenty 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.

The DES procedures are the following:

- **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 (e.g. in a flat file) is another option. Oracle8i allows you to 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, then 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

The following table describes the subprograms discussed in this chapter.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DESEncrypt Procedure&quot; on page 30-5</td>
<td>Generates the encrypted form of the input data.</td>
</tr>
<tr>
<td>&quot;DESDecrypt Procedure&quot; on page 30-7</td>
<td>Generates the decrypted form of the input data.</td>
</tr>
<tr>
<td>&quot;DES3Encrypt Procedure&quot; on page 30-10</td>
<td>Generates the encrypted form of the input data by passing it through the Triple DES (3DES) encryption algorithm.</td>
</tr>
<tr>
<td>&quot;DES3Decrypt Procedure&quot; on page 30-11</td>
<td>Generates the decrypted form of the input data.</td>
</tr>
</tbody>
</table>

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 30–1 and Table 30–2 list the parameters for the DESEncrypt syntax, their modes, types, and descriptions.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>IN</td>
<td>RAW</td>
<td>data to be encrypted</td>
</tr>
<tr>
<td>key</td>
<td>IN</td>
<td>RAW</td>
<td>encryption key</td>
</tr>
</tbody>
</table>
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 30–3 and Table 30–4 list the parameters for the DESDecrypt syntax, their modes, types, and descriptions.

**Table 30–3  DESDecrypt parameters for raw data**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>IN</td>
<td>RAW</td>
<td>Data to be decrypted</td>
</tr>
<tr>
<td>key</td>
<td>IN</td>
<td>RAW</td>
<td>Decryption key</td>
</tr>
<tr>
<td>decrypted_data</td>
<td>OUT</td>
<td>RAW</td>
<td>Decrypted data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>String to be decrypted</td>
</tr>
<tr>
<td>key_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>Decryption key string</td>
</tr>
<tr>
<td>decrypted_string</td>
<td>OUT</td>
<td>VARCHAR2</td>
<td>Decrypted string</td>
</tr>
</tbody>
</table>

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.

```sql
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 testing raw data encryption and decryption');
    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));
    END;
    BEGIN
        dbms_obfuscation_toolkit.DESDecrypt(input => encrypted_raw,
            key => raw_key, decrypted_data => decrypted_raw);
    END;
```

Note: The key length limitation is a requirement of U.S. regulations governing the export of cryptographic products.
SUMMARY OF DBMS_OBFUSCATION SUBPROGRAMS

- 2. Begin testing string data encryption and decryption
- 2. Begin testing string data encryption and decryption
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 30–5 and Table 30–6 list the parameters for the DES3Encrypt syntax, their modes, types, and descriptions.

**Table 30–5  DES3Encrypt parameters for raw data**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>IN</td>
<td>RAW</td>
<td>data to be encrypted</td>
</tr>
<tr>
<td>key</td>
<td>IN</td>
<td>RAW</td>
<td>encryption key</td>
</tr>
<tr>
<td>encrypted_data</td>
<td>OUT</td>
<td>RAW</td>
<td>encrypted data</td>
</tr>
<tr>
<td>which</td>
<td>IN</td>
<td>PLS_INTEGER</td>
<td>If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.</td>
</tr>
</tbody>
</table>

**Table 30–6  DES3Encrypt parameters for string data**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>string to be encrypted</td>
</tr>
<tr>
<td>key_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>encryption key string</td>
</tr>
<tr>
<td>encrypted_string</td>
<td>OUT</td>
<td>VARCHAR2</td>
<td>encrypted string</td>
</tr>
<tr>
<td>which</td>
<td>IN</td>
<td>PLS_INTEGER</td>
<td>If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.</td>
</tr>
</tbody>
</table>
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 30–7 and Table 30–8 list the parameters for the DES3Decrypt syntax, their modes, types, and descriptions.

Table 30–7  DES3Decrypt parameters for raw data

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>IN</td>
<td>RAW</td>
<td>Data to be decrypted</td>
</tr>
<tr>
<td>key</td>
<td>IN</td>
<td>RAW</td>
<td>Decryption key</td>
</tr>
<tr>
<td>decrypted_data</td>
<td>OUT</td>
<td>RAW</td>
<td>Decrypted data</td>
</tr>
<tr>
<td>which</td>
<td>IN</td>
<td>PLS_INTEGER</td>
<td>If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.</td>
</tr>
</tbody>
</table>

Table 30–8  DES3Decrypt parameters for string data

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>String to be decrypted</td>
</tr>
<tr>
<td>key_string</td>
<td>IN</td>
<td>VARCHAR2</td>
<td>Decryption key string</td>
</tr>
<tr>
<td>decrypted_string</td>
<td>OUT</td>
<td>VARCHAR2</td>
<td>Decrypted string</td>
</tr>
<tr>
<td>which</td>
<td>IN</td>
<td>PLS_INTEGER</td>
<td>If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.</td>
</tr>
</tbody>
</table>

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

As stated above, 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.

```sql
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));
    IF UTL_RAW.CAST_TO_VARCHAR2(error_in_input_buffer_length) IS NOT NULL THEN
        RAISE error_in_input_buffer_length;
        dbms_output.put_line('*** DES INPUT BUFFER NOT A MULTIPLE OF 8 BYTES - IGNORING EXCEPTION ***');
    END IF;
    IF UTL_RAW.CAST_TO_VARCHAR2(double_encrypt_not_permitted) IS NOT NULL THEN
        RAISE double_encrypt_not_permitted;
        dbms_output.put_line('*** CANNOT DOUBLE ENCRYPT DATA - IGNORING EXCEPTION ***');
    END IF;
    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 Encryption 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
    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 Encryption 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;

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
**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**

```sql
Function ESTIMATE_CPU_UNITS(elapsed_time NUMBER) RETURN NUMBER;
```

**Parameters**

- `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:

```sql
DECLARE
```

Table 31–1  **DBMS_ODCI Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ESTIMATE_CPU_UNITS Function&quot; on page 31-2</td>
<td>Returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds).</td>
</tr>
</tbody>
</table>
a INTEGER;
BEGIN
  a := DBMS_ODCI.estimate_cpu_units(10);
  DBMS_OUTPUT.PUT_LINE('CPU units = ' || a * 1000);
END;
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 32–1  DBMS_OFFLINE_OG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BEGIN_INSTANTIATION Procedure&quot; on page 32-3</td>
<td>Starts offline instantiation of a master group.</td>
</tr>
<tr>
<td>&quot;BEGIN_LOAD Procedure&quot; on page 32-4</td>
<td>Disables triggers while data is imported to new master site as part of offline instantiation.</td>
</tr>
<tr>
<td>&quot;END_INSTANTIATION Procedure&quot; on page 32-6</td>
<td>Completes offline instantiation of a master group.</td>
</tr>
<tr>
<td>&quot;END_LOAD Procedure&quot; on page 32-7</td>
<td>Re-enables triggers after importing data to new master site as part of offline instantiation.</td>
</tr>
<tr>
<td>&quot;RESUME_SUBSET_OF_MASTERS Procedure&quot; on page 32-9</td>
<td>Resumes replication activity at all existing sites except the new site during offline instantiation of a master group.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
    gname IN VARCHAR2,
    new_site IN VARCHAR2,
    fname IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group that you want to replicate to the new site.</td>
</tr>
<tr>
<td>new_site</td>
<td>The fully qualified database name of the new site to which you want to replicate the replication group.</td>
</tr>
<tr>
<td>fname</td>
<td>This parameter is for internal use only. <strong>Note:</strong> Do not set this parameter unless directed to do so by Oracle Support Services.</td>
</tr>
</tbody>
</table>
BEGIN_LOAD Procedure

Exceptions

Table 32–3 BEGIN_INSTANTIATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group or new master site name.</td>
</tr>
<tr>
<td>dbms_repcat.nonmasterdef</td>
<td>This procedure must be called from the master definition site.</td>
</tr>
<tr>
<td>sitealreadyexists</td>
<td>Specified site is already a master site for this replication group.</td>
</tr>
<tr>
<td>wrongstate</td>
<td>Status of master definition site must be quiesced.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a master group.</td>
</tr>
<tr>
<td>dbms_repcat.missing_flavor</td>
<td>If you receive this exception, contact Oracle Support Services.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_OFFLINE_OG.BEGIN_LOAD (  
gname IN VARCHAR2,  
new_site IN VARCHAR2);
```
Parameters

Table 32–4  BEGIN_LOAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group whose members you are importing.</td>
</tr>
<tr>
<td>new_site</td>
<td>The fully qualified database name of the new site at which you will be importing the replication group members.</td>
</tr>
</tbody>
</table>

Exceptions

Table 32–5  BEGIN_LOAD Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group or new master site name.</td>
</tr>
<tr>
<td>wrongsite</td>
<td>This procedure must be called from the new master site.</td>
</tr>
<tr>
<td>unknownsite</td>
<td>Specified site is not recognized by replication group.</td>
</tr>
<tr>
<td>wrongstate</td>
<td>Status of the new master site must be quiesced.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a master group.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group that you are replicating to the new site.</td>
</tr>
<tr>
<td>new_site</td>
<td>The fully qualified database name of the new site to which you are replicating the replication group.</td>
</tr>
</tbody>
</table>
Summary of DBMS_OFFLINE_OG Subprograms

Exceptions

Table 32–7 END_INSTANTIATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group or new master site name.</td>
</tr>
<tr>
<td>dbms_repcat.nonmasterdef</td>
<td>This procedure must be called from the master definition site.</td>
</tr>
<tr>
<td>unknownsite</td>
<td>Specified site is not recognized by replication group.</td>
</tr>
<tr>
<td>wrongstate</td>
<td>Status of master definition site must be quiesced.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a master group.</td>
</tr>
</tbody>
</table>

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.
**END_LOAD Procedure**

**Syntax**

```sql
DBMS_OFFLINE_OG.END_LOAD (  
gname IN VARCHAR2,  
new_site IN VARCHAR2  
fname IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gname</code></td>
<td>Name of the replication group whose members you have finished importing.</td>
</tr>
<tr>
<td><code>new_site</code></td>
<td>The fully qualified database name of the new site at which you have imported the replication group members.</td>
</tr>
<tr>
<td><code>fname</code></td>
<td>This parameter is for internal use only. <strong>Note:</strong> Do not set this parameter unless directed to do so by Oracle Support Services.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>badargument</code></td>
<td>NULL or empty string for replication group or new master site name.</td>
</tr>
<tr>
<td><code>wrongsite</code></td>
<td>This procedure must be called from the new master site.</td>
</tr>
<tr>
<td><code>unknownsite</code></td>
<td>Specified site is not recognized by replication group.</td>
</tr>
<tr>
<td><code>wrongstate</code></td>
<td>Status of the new master site must be quiesced.</td>
</tr>
<tr>
<td><code>dbms_repcat.missingrepgroup</code></td>
<td><code>gname</code> does not exist as a master group.</td>
</tr>
<tr>
<td><code>dbms_repcat.flavor_noobject</code></td>
<td>If you receive this exception, contact Oracle Support Services.</td>
</tr>
<tr>
<td><code>dbms_repcat.flavor_contains</code></td>
<td>If you receive this exception, contact Oracle Support Services.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
    gname IN VARCHAR2,
    new_site IN VARCHAR2,
    override IN BOOLEAN := false);
```
RESUME_SUBSET_OF_MASTERS Procedure

Parameters

Table 32–10  RESUME_SUBSET_OF_MASTERS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group that you are replicating to the new site.</td>
</tr>
<tr>
<td>new_site</td>
<td>The fully qualified database name of the new site to which you are replicating the replication group.</td>
</tr>
<tr>
<td>override</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 32–11  RESUME_SUBSET_OF_MASTERS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group or new master site name.</td>
</tr>
<tr>
<td>dbms_repcat.nonmasterdef</td>
<td>This procedure must be called from the master definition site.</td>
</tr>
<tr>
<td>unknownsite</td>
<td>Specified site is not recognized by replication group.</td>
</tr>
<tr>
<td>wrongstate</td>
<td>Status of master definition site must be quiesced.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a master group.</td>
</tr>
</tbody>
</table>
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 procedures 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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BEGIN_LOAD Procedure&quot; on page 33-3</td>
<td>Prepares a materialized view site for import of a new materialized view as part of offline instantiation.</td>
</tr>
<tr>
<td>&quot;END_LOAD Procedure&quot; on page 33-5</td>
<td>Completes offline instantiation of a materialized view.</td>
</tr>
</tbody>
</table>
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**

```sql
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);
```
BEGIN_LOAD Procedure

Parameters

Table 33–2  BEGIN_LOAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group for the materialized view that you are creating using offline instantiation.</td>
</tr>
<tr>
<td>sname</td>
<td>Name of the schema for the new materialized view.</td>
</tr>
<tr>
<td>master_site</td>
<td>Fully qualified database name of the materialized view’s master site.</td>
</tr>
<tr>
<td>snapshot_oname</td>
<td>Name of the temporary materialized view created at the master site.</td>
</tr>
<tr>
<td>storage_c</td>
<td>Storage options to use when creating the new materialized view at the materialized view site.</td>
</tr>
<tr>
<td>comment</td>
<td>User comment.</td>
</tr>
<tr>
<td>min_communication</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 33–3  BEGIN_LOAD Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group, schema, master site, or materialized view name.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a replication group.</td>
</tr>
<tr>
<td>missingremotemview</td>
<td>Could not locate specified materialized view at specified master site.</td>
</tr>
<tr>
<td>dbms_repcat.missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>mviewtabmismatch</td>
<td>Base table name of the materialized view at the master and materialized view do not match.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
  gname IN VARCHAR2,
  sname IN VARCHAR2,
  snapshot_oname IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group for the materialized view that you are creating using offline instantiation.</td>
</tr>
<tr>
<td>sname</td>
<td>Name of the schema for the new materialized view.</td>
</tr>
<tr>
<td>snapshot_oname</td>
<td>Name of the materialized view.</td>
</tr>
</tbody>
</table>
## Exceptions

**Table 33–5  END_LOAD Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badargument</td>
<td>NULL or empty string for replication group, schema, or materialized view name.</td>
</tr>
<tr>
<td>dbms_repcat.missingrepgroup</td>
<td>gname does not exist as a replication group.</td>
</tr>
<tr>
<td>dbms_repcat.nonnview</td>
<td>This procedure must be called from the materialized view site.</td>
</tr>
</tbody>
</table>
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 pre-process collected workload statistics into default V-tables in the user’s schema.

Error Messages

Table 34–1 lists basic DBMS_OLAP error messages.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-30442</td>
<td>Cannot find the definition for filter &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>ORA-30443</td>
<td>Definition for filter &lt;NUMBER&gt;’s item &lt;NUMBER&gt; is invalid</td>
</tr>
<tr>
<td>ORA-30444</td>
<td>Rewrite terminated by the SQL Analyzer</td>
</tr>
<tr>
<td>ORA-30445</td>
<td>Workload queries not found</td>
</tr>
<tr>
<td>ORA-30446</td>
<td>Valid workload queries not found</td>
</tr>
<tr>
<td>ORA-30447</td>
<td>internal data for run number &lt;NUMBER&gt; is inconsistent</td>
</tr>
<tr>
<td>ORA-30448</td>
<td>Internal data of the Advisor repository is inconsistent</td>
</tr>
<tr>
<td>ORA-30449</td>
<td>Syntax error in parameter &lt;NUMBER&gt;</td>
</tr>
</tbody>
</table>
Table 34–1  DBMS OLAP Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-30465</td>
<td>Supplied run_id is not valid: &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>ORA-30466</td>
<td>Cannot find the specified workload &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>ORA-30477</td>
<td>The input select_clause is incorrectly specified</td>
</tr>
<tr>
<td>ORA-30478</td>
<td>Specified dimension does not exist</td>
</tr>
<tr>
<td>ORA-30479</td>
<td>Summary Advisor error QSM message with more details</td>
</tr>
<tr>
<td>QSM-00501</td>
<td>Unable to initialize Summary Advisor environment</td>
</tr>
<tr>
<td>QSM-00502</td>
<td>OCI error</td>
</tr>
<tr>
<td>QSM-00503</td>
<td>Out of memory</td>
</tr>
<tr>
<td>QSM-00504</td>
<td>Internal error</td>
</tr>
<tr>
<td>QSM-00505</td>
<td>Syntax error in &lt;parse_entity_name&gt;-&lt;error_description&gt;</td>
</tr>
<tr>
<td>QSM-00506</td>
<td>No fact-tables could be found</td>
</tr>
<tr>
<td>QSM-00507</td>
<td>No dimensions could be found</td>
</tr>
<tr>
<td>QSM-00508</td>
<td>Statistics missing on tables/columns</td>
</tr>
<tr>
<td>QSM-00509</td>
<td>Invalid parameter - &lt;parameter_name&gt;</td>
</tr>
<tr>
<td>QSM-00510</td>
<td>Statistics missing on summaries</td>
</tr>
<tr>
<td>QSM-00511</td>
<td>Invalid fact-tables specified in fact-filter</td>
</tr>
<tr>
<td>QSM-00512</td>
<td>Invalid summaries specified in the retention-list</td>
</tr>
<tr>
<td>QSM-00513</td>
<td>One or more of the workload tables is missing</td>
</tr>
<tr>
<td>QSM-00550</td>
<td>The filter item type &lt;NAME&gt; is missing the required data</td>
</tr>
<tr>
<td>QSM-00551</td>
<td>The file &lt;NAME&gt; was not found</td>
</tr>
<tr>
<td>QSM-00552</td>
<td>The workload source was not defined or was not recognized</td>
</tr>
<tr>
<td>QSM-00553</td>
<td>The string value for filter item &lt;NAME&gt; has a maximum length of &lt;NUMBER&gt; characters</td>
</tr>
<tr>
<td>QSM-00554</td>
<td>The required table name was not provided</td>
</tr>
<tr>
<td>QSM-00555</td>
<td>The table &lt;NAME&gt; can not be accessed or does not exist</td>
</tr>
<tr>
<td>QSM-00556</td>
<td>The file &lt;NAME&gt; could not be opened</td>
</tr>
<tr>
<td>QSM-00557</td>
<td>The owner &lt;NAME&gt; can not be accessed or does not exist</td>
</tr>
</tbody>
</table>
### Table 34–1  DBMS_OLAP Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSM-00558</td>
<td>An error occurred while reading file <code>&lt;NAME&gt;</code></td>
</tr>
<tr>
<td>QSM-00559</td>
<td>A workload already exists for the specified collection ID</td>
</tr>
<tr>
<td>QSM-00560</td>
<td>The character <code>&lt;NAME&gt;</code> is invalid at line <code>&lt;LINE_NUMBER&gt;</code>, column <code>&lt;COLUMN_NUMBER&gt;</code></td>
</tr>
<tr>
<td>QSM-00561</td>
<td>Found <code>&lt;TOKEN&gt;</code> at line <code>&lt;NUMBER&gt;</code>, column <code>&lt;NUMBER&gt;</code>. Expecting 1 of the following items: <code>&lt;ITEMS&gt;</code></td>
</tr>
<tr>
<td>QSM-00562</td>
<td>The requested Advisor task was not found</td>
</tr>
<tr>
<td>QSM-00563</td>
<td>Found <code>&lt;TOKEN&gt;</code> at line <code>&lt;NUMBER&gt;</code>, column <code>&lt;NUMBER&gt;</code> of file <code>&lt;NAME&gt;</code>. Expecting 1 of the following items: <code>&lt;ITEMS&gt;</code></td>
</tr>
<tr>
<td>QSM-00564</td>
<td>An internal lexical error occurred: <code>&lt;Additional error text&gt;</code></td>
</tr>
<tr>
<td>QSM-00565</td>
<td>The <code>&lt;NAME&gt;</code> was not found while validating the <code>&lt;TABLE or COLUMN&gt;</code> at line <code>&lt;NUMBER&gt;</code>, column <code>&lt;NUMBER&gt;</code></td>
</tr>
<tr>
<td>QSM-00566</td>
<td>The <code>&lt;TOKEN&gt;</code> is ambiguous while validating the <code>&lt;TABLE or COLUMN&gt;</code> at line <code>&lt;NUMBER&gt;</code>, column <code>&lt;NUMBER&gt;</code></td>
</tr>
<tr>
<td>QSM-00567</td>
<td>A runtime error occurred: <code>&lt;Additional error text&gt;</code></td>
</tr>
<tr>
<td>QSM-00568</td>
<td>The end-of-file was encountered</td>
</tr>
<tr>
<td>QSM-00569</td>
<td>The required column <code>&lt;NAME&gt;</code> was not found in table <code>&lt;NAME&gt;</code></td>
</tr>
<tr>
<td>QSM-00570</td>
<td>The job has ended in error. Status changes are not permitted</td>
</tr>
<tr>
<td>QSM-00571</td>
<td>The job has already completed. Status changes are unnecessary</td>
</tr>
<tr>
<td>QSM-00572</td>
<td>No repository connection has been established</td>
</tr>
<tr>
<td>QSM-00573</td>
<td>The date <code>&lt;VALUE&gt;</code> must be in the form 'DD/MM/YYYY HH24:MI:SS'</td>
</tr>
<tr>
<td>QSM-00574</td>
<td>The file <code>&lt;NAME&gt;</code> could not be accessed due to a security violation</td>
</tr>
<tr>
<td>QSM-00575</td>
<td>The string <code>&lt;VALUE&gt;</code> can not be converted to a number</td>
</tr>
<tr>
<td>QSM-00576</td>
<td>A usable Oracle Trace collection was not found in schema <code>&lt;NAME&gt;</code></td>
</tr>
<tr>
<td>QSM-00577</td>
<td>The current operation was cancelled by the user</td>
</tr>
<tr>
<td>QSM-00578</td>
<td>A temporary file cannot be created using the specification <code>&lt;FILE_NAME&gt;</code></td>
</tr>
<tr>
<td>QSM-00579</td>
<td>The job has already completed. Cancellation is unnecessary</td>
</tr>
<tr>
<td>QSM-00580</td>
<td>The job has ended in error. Cancellation is not permitted</td>
</tr>
<tr>
<td>QSM-00581</td>
<td>Internal error: <code>&lt;Additional error text&gt;</code></td>
</tr>
</tbody>
</table>
### Table 34–1  DBMS_OLAP Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSM-00582</td>
<td>A database error has occurred. &lt;Additional error text&gt;</td>
</tr>
<tr>
<td>QSM-00583</td>
<td>The filter item type &lt;NAME&gt; is invalid</td>
</tr>
<tr>
<td>QSM-00584</td>
<td>The SQL cache is not accessible by user &lt;NAME&gt;</td>
</tr>
<tr>
<td>QSM-00585</td>
<td>The workload was not found for collection ID &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>QSM-00586</td>
<td>The filter was not found for filter ID &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>QSM-00587</td>
<td>The analysis data was not found for run ID &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>QSM-00588</td>
<td>The current user does not have the privilege to access the requested workload, which is owned by user &lt;NAME&gt;</td>
</tr>
<tr>
<td>QSM-00589</td>
<td>The current user does not have the privilege to access the requested workload filter, which is owned by user &lt;NAME&gt;</td>
</tr>
<tr>
<td>QSM-00590</td>
<td>The current user does not have the privilege to access the requested Advisor items, which are owned by user &lt;NAME&gt;</td>
</tr>
<tr>
<td>QSM-00591</td>
<td>The specified report style &lt;NAME&gt; was not found</td>
</tr>
<tr>
<td>QSM-00592</td>
<td>The specified report field &lt;NAME&gt; already exists</td>
</tr>
<tr>
<td>QSM-00593</td>
<td>The specified report field &lt;NAME&gt; was not found</td>
</tr>
<tr>
<td>QSM-00594</td>
<td>The specified ID number is already being used by another user</td>
</tr>
<tr>
<td>QSM-00595</td>
<td>The specified ID number is being used by an Advisor &lt;NAME&gt; object and can not be used for this operation</td>
</tr>
<tr>
<td>QSM-00596</td>
<td>A specified ID number cannot be NULL or zero</td>
</tr>
<tr>
<td>QSM-00597</td>
<td>Found &lt;TOKEN&gt; at line &lt;NUMBER&gt;, column &lt;NUMBER&gt;</td>
</tr>
<tr>
<td>QSM-00598</td>
<td>The minimum range value for filter item &lt;NAME&gt; is greater than the maximum range value</td>
</tr>
<tr>
<td>QSM-00599</td>
<td>The supplied workload filter contains items that are unsupported for the requested workload operation: &lt;OPERATION&gt;</td>
</tr>
<tr>
<td>QSM-00602</td>
<td>The ID &lt;NUMBER&gt; is not a valid Summary Advisor run or collection ID for the current user</td>
</tr>
<tr>
<td>QSM-00601</td>
<td>The flags value of &lt;NUMBER&gt; for the Summary Advisor detail report is invalid</td>
</tr>
</tbody>
</table>
Summary of DBMS_OLAP Subprograms

Table 34–2 lists the subprograms available with DBMS_OLAP.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_FILTER_ITEM Procedure&quot; on page 34-7</td>
<td>Filters the contents being used during the recommendation process.</td>
</tr>
<tr>
<td>&quot;CREATE_ID Procedure&quot; on page 34-9</td>
<td>Generates an internal ID used by a new workload collection, a new filter, or a new advisor run.</td>
</tr>
<tr>
<td>&quot;ESTIMATE_MVIEW_SIZE Procedure&quot; on page 34-9</td>
<td>Estimates the size of a materialized view that you might create, in bytes and rows.</td>
</tr>
<tr>
<td>&quot;EVALUATE_MVIEW_STRATEGY Procedure&quot; on page 34-10</td>
<td>Measures the utilization of each existing materialized view.</td>
</tr>
<tr>
<td>&quot;GENERATE_MVIEW_REPORT Procedure&quot; on page 34-10</td>
<td>Generates an HTML-based report on the given Advisor run.</td>
</tr>
<tr>
<td>&quot;GENERATE_MVIEW_SCRIPT Procedure&quot; on page 34-11</td>
<td>Generates a simple script containing the SQL commands to implement Summary Advisor recommendations.</td>
</tr>
<tr>
<td>&quot;LOAD_WORKLOAD_CACHE Procedure&quot; on page 34-12</td>
<td>Obtains a SQL cache workload.</td>
</tr>
<tr>
<td>&quot;LOAD_WORKLOAD_TRACE Procedure&quot; on page 34-13</td>
<td>Loads a workload collected by Oracle Trace.</td>
</tr>
<tr>
<td>&quot;LOAD_WORKLOAD_USER Procedure&quot; on page 34-14</td>
<td>Loads a user-defined workload.</td>
</tr>
<tr>
<td>&quot;PURGE_FILTER Procedure&quot; on page 34-15</td>
<td>Deletes a specific filter or all filters.</td>
</tr>
<tr>
<td>&quot;PURGE_RESULTS Procedure&quot; on page 34-16</td>
<td>Removes all results or those for a specific run.</td>
</tr>
<tr>
<td>&quot;PURGE_WORKLOAD Procedure&quot; on page 34-16</td>
<td>Deletes all workloads or a specific collection.</td>
</tr>
<tr>
<td>&quot;RECOMMEND_MVIEW_STRATEGY Procedure&quot; on page 34-17</td>
<td>Generates a set of recommendations about which materialized views should be created, retained, or dropped.</td>
</tr>
<tr>
<td>&quot;SET_CANCELED Procedure&quot; on page 34-18</td>
<td>Stops the Advisor if it takes too long returning results.</td>
</tr>
</tbody>
</table>
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

```sql
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 34–3  ADD_FILTER_ITEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter_id</td>
<td>NUMBER</td>
<td>An ID that uniquely describes the filter. It is generated by the DBMS_OLAP.CREATE_ID procedure</td>
</tr>
</tbody>
</table>
ADD_FILTER_ITEM Procedure

Table 34–3  ADD_FILTER_ITEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
</table>
| filter_name    | VARCHAR2 | APPLICATION
String-workload’s application column. An example of how to load a SQL Cache workload is shown below.
BASETABLE
String-base tables referenced by workload queries. Name must be fully qualified including owner and table name i.e. 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.
TRACENAME
String-list of oracle trace collection names. Only used by a Trace Workload

| string_list    | VARCHAR2 | A comma-separated 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 |
| 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 |
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

```sql
CALL DBMS_OLAP.CREATE_ID (
    id OUT NUMBER);
```

Table 34–4 CREATE_ID Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>NUMBER</td>
<td>The unique identifier that can be used to identify a filter, a workload, or an Advisor run</td>
</tr>
</tbody>
</table>

ESTIMATE_MVIEW_SIZE Procedure

This estimates the size of a materialized view that you might create, in bytes and number of rows.

Syntax

```sql
DBMS_OLAP.ESTIMATE_MVIEW_SIZE (
    stmt_id IN VARCHAR2,
    select_clause IN VARCHAR2,
    num_rows OUT NUMBER,
    num_bytes OUT NUMBER);
```

Parameters

Table 34–5 ESTIMATE_MVIEW_SIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt_id</td>
<td>NUMBER</td>
<td>Arbitrary string used to identify the statement in an EXPLAIN PLAN.</td>
</tr>
</tbody>
</table>
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 34-21

Syntax

```
DBMS_OLAP.EVALUATE_MVIEW_STRATEGY (run_id IN NUMBER, workload_id IN NUMBER, filter_id IN NUMBER);
```

Parameters

Table 34–5  ESTIMATE_MVIEW_SIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_clause</td>
<td>STRING</td>
<td>The SELECT statement to be analyzed.</td>
</tr>
<tr>
<td>num_rows</td>
<td>NUMBER</td>
<td>Estimated cardinality.</td>
</tr>
<tr>
<td>num_bytes</td>
<td>NUMBER</td>
<td>Estimated number of bytes.</td>
</tr>
</tbody>
</table>

GENERATE_MVIEW_REPORT Procedure

Generates an HTML-based report on the given Advisor run.

Table 34–6  EVALUATE_MVIEW_STRATEGY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_id</td>
<td>NUMBER</td>
<td>An ID generated by the DBMS_OLAP.CREATE_ID procedure to identify results of a run</td>
</tr>
<tr>
<td>workload_id</td>
<td>NUMBER</td>
<td>An optional workload ID that maps to a workload in the current repository. Use the parameter DBMS_OLAP.WORKLOAD_ALL to choose all workloads</td>
</tr>
<tr>
<td>filter_id</td>
<td>NUMBER</td>
<td>Specify filter for the workload to be used. The value DBMS_OLAP.FILTER_NONE indicates no filtering</td>
</tr>
</tbody>
</table>

Table 34–5  ESTIMATE_MVIEW_SIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_clause</td>
<td>STRING</td>
<td>The SELECT statement to be analyzed.</td>
</tr>
<tr>
<td>num_rows</td>
<td>NUMBER</td>
<td>Estimated cardinality.</td>
</tr>
<tr>
<td>num_bytes</td>
<td>NUMBER</td>
<td>Estimated number of bytes.</td>
</tr>
</tbody>
</table>
Syntax

DBMS_OLAP.GENERATE_MVIEW_REPORT (  
filename IN VARCHAR2,  
id IN NUMBER,  
flags IN NUMBER);

Table 34–7 GENERATE_MVIEW_REPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>VARCHAR2</td>
<td>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.</td>
</tr>
<tr>
<td>id</td>
<td>NUMBER</td>
<td>An ID that identifies an advisor run. Or use the parameter DBMS_OLAP.RUNID_ALL to indicate all advisor runs should be reported.</td>
</tr>
</tbody>
</table>
| 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,  
flags IN NUMBER);
LOAD_WORKLOAD_CACHE Procedure

Loads a SQL cache workload.

Syntax

```sql
DBMS_OLAP.LOAD_WORKLOAD_CACHE(
    workload_id IN NUMBER,
    flags IN NUMBER,
    filter_id IN NUMBER,
    application IN VARCHAR2,
    priority IN NUMBER);
```

Table 34–8 GENERATE_MVIEW_SCRIPT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>VARCHAR2</td>
<td>Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the &quot;Security and Performance&quot; section of the Java Developer’s Guide for more information on file permissions</td>
</tr>
<tr>
<td>id</td>
<td>NUMBER</td>
<td>An ID that identifies an advisor run. The parameter DBMS_OLAP.RUNID_ALL indicates all advisor runs should be reported.</td>
</tr>
<tr>
<td>tspace</td>
<td>VARCHAR2</td>
<td>Optional tablespace name to use when creating materialized views.</td>
</tr>
</tbody>
</table>

Table 34–9 LOAD_WORKLOAD_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_id</td>
<td>NUMBER</td>
<td>Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the &quot;Security and Performance&quot; section of the Java Developer’s Guide for more information on file permissions</td>
</tr>
</tbody>
</table>
Summary of DBMS_OLAP Subprograms

LOAD_WORKLOAD_TRACE Procedure

Loads an Oracle Trace workload.

Syntax

```sql
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 34–9 LOAD_WORKLOAD_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>NUMBER</td>
<td>The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS_OLAP.WORKLOAD_APPEND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS_OLAP.WORKLOAD_NEW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: the flags have the same behavior irrespective of the LOAD_WORKLOAD operation</td>
</tr>
<tr>
<td>filter_id</td>
<td>NUMBER</td>
<td>Specify filter for the workload to be loaded</td>
</tr>
<tr>
<td>application</td>
<td>VARCHAR2</td>
<td>The default business application name. This value will be used for a query if one is not found in the target workload</td>
</tr>
<tr>
<td>priority</td>
<td>NUMBER</td>
<td>The default business priority to be assigned to every query in the target workload</td>
</tr>
</tbody>
</table>
A user-defined workload is loaded using the procedure `LOAD_WORKLOAD_USER`.

**Syntax**

```sql
DBMS_OLAP.LOAD_WORKLOAD_USER (workload_id IN NUMBER,
flag IN NUMBER,
filter_id IN NUMBER,
application VARCHAR2,
priority NUMBER
);```

**Table 34–10  LOAD_WORKLOAD_TRACE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectionid</td>
<td>NUMBER</td>
<td>Fully qualified output file name to receive HTML data. Note that the Oracle server restricts file access within Oracle stored procedures. See the &quot;Security and Performance” section of the Java Developer’s Guide for more information on file permission.</td>
</tr>
<tr>
<td>flags</td>
<td>NUMBER</td>
<td>DBMS_OLAP.WORKLOAD_OVERWRITE The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID.</td>
</tr>
<tr>
<td>filter_id</td>
<td>NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>application</td>
<td>VARCHAR2</td>
<td>The default business application name. This value will be used for a query if one is not found in the target workload.</td>
</tr>
<tr>
<td>priority</td>
<td>NUMBER</td>
<td>The default business priority to be assigned to every query in the target workload.</td>
</tr>
<tr>
<td>owner_name</td>
<td>VARCHAR2</td>
<td>The schema that contains the Oracle Trace data. If omitted, the current user will be used.</td>
</tr>
</tbody>
</table>
owner_name  IN VARCHAR2,
table_name  IN VARCHAR2);

Table 34–11  LOAD_WORKLOAD_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_id</td>
<td>NUMBER</td>
<td>The required id that was returned by the DBMS_OLAP.CREATE_ID call</td>
</tr>
<tr>
<td>flags</td>
<td>NUMBER</td>
<td>DBMS_OLAP.WORKLOAD_OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The load routine will explicitly remove any existing queries from the workload that are owned by the specified collection ID DBMS_OLAP.WORKLOAD_APPEND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td>filter_id</td>
<td>NUMBER</td>
<td>Specify filter for the workload to be loaded</td>
</tr>
<tr>
<td>owner_name</td>
<td>VARCHAR2</td>
<td>The schema that contains the user supplied table or view</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR2</td>
<td>The table or view name containing valid workload data</td>
</tr>
</tbody>
</table>

PURGE_FILTER Procedure

A filter can be removed at anytime by calling the procedure PURGE_FILTER which is described below. You can delete a specific filter or all filters.

Syntax

DBMS_OLAP.PURGE_FILTER (filter_id IN NUMBER);
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 34-21. 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

```sql
DBMS_OLAP.PURGE_RESULTS (
   run_id IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_id</td>
<td>NUMBER</td>
<td>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</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_OLAP.PURGE_WORKLOAD (
   workload_id IN NUMBER);
```
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 34-21

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,
)
SET_CANCELLED Procedure

```sql
fact_table_filter IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_id</td>
<td>An ID generated by the DBMS_OLAP.CREATE_ID procedure to uniquely identify results of a run</td>
</tr>
<tr>
<td>workload_id</td>
<td>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</td>
</tr>
<tr>
<td>filter_id</td>
<td>An optional filter ID that maps to a set of user-supplied filter items. Use the parameter DBMS_OLAP.FILTER_NONE to avoid filtering</td>
</tr>
<tr>
<td>storage_in_bytes</td>
<td>Maximum storage, in bytes, that can be used for storing materialized views. This number must be non-negative.</td>
</tr>
<tr>
<td>retention_pct</td>
<td>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 (e.g. non-dimensional materialized views) are always retained.</td>
</tr>
<tr>
<td>retention_list</td>
<td>Comma-separated list of materialized view table names. A drop recommendation is not made for any materialized view that appears in this list.</td>
</tr>
<tr>
<td>fact_table_filter</td>
<td>Optional list of fact tables used to filter real or ideal workload</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_OLAP Subprograms

Syntax

DBMS_OLAP.SET_CANCELLED (  
    run_id           IN  NUMBER);  

Table 34–16  DBMS_OLAP.SET_CANCELLED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_id</td>
<td>NUMBER</td>
<td>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</td>
</tr>
</tbody>
</table>

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 34-21

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 34–17  VALIDATE_DIMENSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_name</td>
<td>Name of the dimension to analyze.</td>
</tr>
<tr>
<td>dimension_owner</td>
<td>Name of the dimension owner.</td>
</tr>
</tbody>
</table>
VALIDATE_WORKLOAD_CACHE Procedure

This procedure validates the SQL Cache workload before performing load operations.

Syntax

```sql
DBMS_OLAP.VALIDATE_WORKLOAD_CACHE (  
    valid OUT NUMBER,  
    error OUT VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incremental</td>
<td>If TRUE, then tests are performed only for the rows specified in the sumdelta$ table for tables of this dimension; otherwise, check all rows.</td>
</tr>
<tr>
<td>check_nulls</td>
<td>If TRUE, then all level columns are verified to be non-NULL; otherwise, this check is omitted. Specify FALSE when non-nullness is guaranteed by other means, such as NOT NULL constraints.</td>
</tr>
<tr>
<td>run_id</td>
<td>An ID generated by the DBMS_OLAP.CREATE_ID procedure to identify a run.</td>
</tr>
</tbody>
</table>

VALIDATE_WORKLOAD_TRACE Procedure

This procedure validates the Oracle Trace workload before performing load operations.

Syntax

```sql
DBMS_OLAP.VALIDATE_WORKLOAD_TRACE (  
    owner_name IN VARCHAR2,
    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 non-NULL; otherwise, this check is omitted. Specify FALSE when non-nullness 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.  
```

Table 34–18 VALIDATE_WORKLOAD_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid</td>
<td>Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID to indicate whether a workload is valid.</td>
</tr>
<tr>
<td>error</td>
<td>VARCHAR2, return error set</td>
</tr>
</tbody>
</table>

34-20  Supplied PL/SQL Packages and Types Reference
Parameters

**Table 34–19 VALIDATE_WORKLOAD_TRACE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Owner of the trace workload table</td>
</tr>
<tr>
<td>valid</td>
<td>Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.</td>
</tr>
<tr>
<td>error</td>
<td>VARCHAR2, return error text</td>
</tr>
</tbody>
</table>

**VALIDATE_WORKLOAD_USER Procedure**

This procedure validates the user-supplied workload before performing load operations.

**Syntax**

```sql
DBMS_OLAP.VALIDATE_WORKLOAD_USER (
    owner_name  IN VARCHAR2,
    table_name  IN VARCHAR2,
    valid       OUT NUMBER,
    error       OUT VARCHAR2);
```

**Parameters**

**Table 34–20 VALIDATE_WORKLOAD_USER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Owner of the user workload table</td>
</tr>
<tr>
<td>table_name</td>
<td>User workload table name</td>
</tr>
<tr>
<td>valid</td>
<td>Return DBMS_OLAP.VALID or DBMS_OLAP.INVALID Indicate whether a workload is valid.</td>
</tr>
<tr>
<td>error</td>
<td>VARCHAR2, return error set</td>
</tr>
</tbody>
</table>
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 34–21  SYSTEM.MVIEW_EVALUATIONS**

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Run id identifying a unique advisor call</td>
</tr>
<tr>
<td>MVIEW_OWNER</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Owner of materialized view</td>
</tr>
<tr>
<td>MVIEW_NAME</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Name of an exiting materialized view in this database</td>
</tr>
<tr>
<td>RANK</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Rank of this materialized view in descending order of benefit_to_cost_ratio</td>
</tr>
<tr>
<td>STORAGE_IN_BYTES</td>
<td></td>
<td>NUMBER</td>
<td>Size of the materialized view in bytes</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td></td>
<td>NUMBER</td>
<td>Number of times this materialized view appears in the workload</td>
</tr>
<tr>
<td>CUMULATIVE_BENEFIT</td>
<td></td>
<td>NUMBER</td>
<td>The cumulative benefit of the materialized view</td>
</tr>
<tr>
<td>BENEFIT_TO_COST_RATIO</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>The ratio of cumulative_benefit to storage_in_bytes</td>
</tr>
</tbody>
</table>

### SYSTEM.MVIEW_EXCEPTIONS

**Table 34–22  SYSTEM.MVIEW_EXCEPTIONS**

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNID</td>
<td></td>
<td>NUMBER</td>
<td>Run id identifying a unique advisor call</td>
</tr>
<tr>
<td>OWNER</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Owner name</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Table name</td>
</tr>
</tbody>
</table>
### Table 34–22  SYSTEM.MVIEW_EXCEPTIONS

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMENSION_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
<td>Dimension name</td>
</tr>
<tr>
<td>RELATIONSHIP</td>
<td></td>
<td>VARCHAR2(11)</td>
<td>Violated relation name</td>
</tr>
<tr>
<td>BAD_ROWID</td>
<td></td>
<td>ROWID</td>
<td>Location of offending entry</td>
</tr>
</tbody>
</table>

### SYSTEM.MVIEW_FILTER

### Table 34–23  SYSTEM.MVIEW_FILTER

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTERID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Unique number used to identify the operation that used this filter</td>
</tr>
<tr>
<td>SUBFILTERNUM</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>A unique id number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table</td>
</tr>
<tr>
<td>SUBFILTERTYPE</td>
<td></td>
<td>VARCHAR2(12)</td>
<td>Filter item number</td>
</tr>
<tr>
<td>STR_VALUE</td>
<td></td>
<td>VARCHAR2(1028)</td>
<td>String attribute for items that require strings</td>
</tr>
<tr>
<td>NUM_VALUE1</td>
<td></td>
<td>NUMBER</td>
<td>Numeric low for items that require numbers</td>
</tr>
<tr>
<td>NUM_VALUE2</td>
<td></td>
<td>NUMBER</td>
<td>Numeric high for items that require numbers</td>
</tr>
<tr>
<td>DATE_VALUE1</td>
<td></td>
<td>DATE</td>
<td>Date low for items that require dates</td>
</tr>
<tr>
<td>DATE_VALUE2</td>
<td></td>
<td>DATE</td>
<td>Date high for items that require dates</td>
</tr>
</tbody>
</table>
SYSTEM.MVIEW_FILTERINSTANCE

Table 34–24 SYSTEM.MVIEW_FILTER

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Unique number used to identify the operation that used this filter</td>
</tr>
<tr>
<td>FILTERID</td>
<td></td>
<td>NUMBER</td>
<td>A unique id number that groups all filter items together. A corresponding filter header record can be found in the MVIEW_LOG table</td>
</tr>
<tr>
<td>SUBFILTERNUM</td>
<td></td>
<td>NUMBER</td>
<td>Filter item number</td>
</tr>
<tr>
<td>SUBFILTERTYPE</td>
<td></td>
<td>VARCHAR2(12)</td>
<td>Filter item type</td>
</tr>
<tr>
<td>STR_VALUE</td>
<td></td>
<td>VARCHAR2(1028)</td>
<td>String attribute for items that require strings</td>
</tr>
<tr>
<td>NUM_VALUE1</td>
<td></td>
<td>NUMBER</td>
<td>Numeric low for items that require numbers</td>
</tr>
<tr>
<td>NUM_VALUE2</td>
<td></td>
<td>NUMBER</td>
<td>Numeric high for items that require numbers</td>
</tr>
<tr>
<td>DATE_VALUE1</td>
<td></td>
<td>DATE</td>
<td>Date low for items that require dates</td>
</tr>
<tr>
<td>DATE_VALUE2</td>
<td></td>
<td>DATE</td>
<td>Date high for items that require dates</td>
</tr>
</tbody>
</table>

SYSTEM.MVIEW_LOG

Table 34–25 SYSTEM.MVIEW_LOG

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Unique number used to identify the table entry. The number must be created using the CREATE_ID routine</td>
</tr>
<tr>
<td>FILTERID</td>
<td></td>
<td>NUMBER</td>
<td>Optional filter id. Zero indicates no user-supplied filter has been applied to the operation</td>
</tr>
<tr>
<td>RUN_BEGIN</td>
<td></td>
<td>DATE</td>
<td>Date at which the operation began</td>
</tr>
<tr>
<td>RUN_END</td>
<td></td>
<td>DATE</td>
<td>Date at which the operation ended</td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
<td>VARCHAR2(11)</td>
<td>A name that identifies the type of operation</td>
</tr>
<tr>
<td>STATUS</td>
<td></td>
<td>VARCHAR2(11)</td>
<td>The current operational status</td>
</tr>
</tbody>
</table>
### Table 34–25  **SYSTEM.MVIEW_LOG**

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td></td>
<td>VARCHAR2 (2000)</td>
<td>Informational message indicating current operation or condition</td>
</tr>
<tr>
<td>COMPLETED</td>
<td></td>
<td>NUMBER</td>
<td>Number of steps completed by operation</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>NUMBER</td>
<td>Total number steps to be performed</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td></td>
<td>VARCHAR2 (20)</td>
<td>Oracle error code in the event of an error</td>
</tr>
</tbody>
</table>

### SYSTEM.MVIEW_RECOMMENDATIONS

### Table 34–26  **SYSTEM.MVIEW_RECOMMENDATIONS**

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNID</td>
<td></td>
<td>NUMBER</td>
<td>Run id identifying a unique advisor call</td>
</tr>
<tr>
<td>ALL_TABLES</td>
<td></td>
<td>VARCHAR2 (2000)</td>
<td>A comma-separated list of fully qualified table names for structured recommendations</td>
</tr>
<tr>
<td>FACT_TABLES</td>
<td></td>
<td>VARCHAR2 (1000)</td>
<td>A comma-separated list of grouping levels, if any, for structured recommendation</td>
</tr>
<tr>
<td>GROUPING_LEVELS</td>
<td></td>
<td>VARCHAR2 (2000)</td>
<td></td>
</tr>
<tr>
<td>QUERY_TEXT</td>
<td></td>
<td>LONG</td>
<td>Query text of materialized view if RECOMMENDED_ACTION is CREATE; null otherwise</td>
</tr>
<tr>
<td>RECOMMENDATION_ NUMBER</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Unique identifier for this recommendation</td>
</tr>
<tr>
<td>RECOMMENDED_ACTION</td>
<td></td>
<td>VARCHAR2 (6)</td>
<td>CREATE, RETAIN, or DROP, Retain, Create, or Drop</td>
</tr>
<tr>
<td>MVIEW_OWNER</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Owner of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise</td>
</tr>
<tr>
<td>MVIEW_NAME</td>
<td></td>
<td>VARCHAR2 (30)</td>
<td>Name of the materialized view if RECOMMENDED_ACTION is RETAIN or DROP; null otherwise</td>
</tr>
</tbody>
</table>
Table 34–26  SYSTEM.MVIEW_RECOMMENDATIONS

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORAGE_IN_BYTES</td>
<td></td>
<td>NUMBER</td>
<td>Actual or estimated storage in bytes</td>
</tr>
<tr>
<td>PCT_PERFORMANCE_GAIN</td>
<td></td>
<td>NUMBER</td>
<td>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</td>
</tr>
<tr>
<td>BENEFIT_TO_COST_RATIO</td>
<td>NOT NULL</td>
<td>NUMBER</td>
<td>Ratio of the incremental improvement in performance to the size of the materialized view in bytes, or NULL if unknown</td>
</tr>
</tbody>
</table>

**SYSTEM.MVIEW_WORKLOAD**

Table 34–27  SYSTEM.MVIEW_WORKLOAD

<table>
<thead>
<tr>
<th>Column</th>
<th>NULL?</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION</td>
<td>VARCHAR2 (30)</td>
<td></td>
<td>Optional application name for the query</td>
</tr>
<tr>
<td>CARDINALITY</td>
<td>NUMBER</td>
<td></td>
<td>Total cardinality of all of tables in query</td>
</tr>
<tr>
<td>WORKLOADID</td>
<td>NUMBER</td>
<td></td>
<td>Workload id identifying a unique sampling</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td>NUMBER</td>
<td></td>
<td>Number of times query executed</td>
</tr>
<tr>
<td>IMPORT_TIME</td>
<td>DATE</td>
<td></td>
<td>Date at which item was collected</td>
</tr>
<tr>
<td>LASTUSE</td>
<td>DATE</td>
<td></td>
<td>Last date of execution</td>
</tr>
<tr>
<td>OWNER</td>
<td>VARCHAR2 (30)</td>
<td></td>
<td>User who last executed query</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>NUMBER</td>
<td></td>
<td>User-supplied ranking of query</td>
</tr>
<tr>
<td>QUERY</td>
<td>LONG</td>
<td></td>
<td>Query text</td>
</tr>
<tr>
<td>QUERYID</td>
<td>NUMBER</td>
<td></td>
<td>Id number identifying a unique query</td>
</tr>
<tr>
<td>RESPONSETIME</td>
<td>NUMBER</td>
<td></td>
<td>Execution time in seconds</td>
</tr>
<tr>
<td>RESULTSIZE</td>
<td>NUMBER</td>
<td></td>
<td>Total bytes selected by the query</td>
</tr>
</tbody>
</table>
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 values are taken from v$session.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>Session ID.</td>
</tr>
<tr>
<td>serial#</td>
<td>Session serial number.</td>
</tr>
<tr>
<td>on_off</td>
<td>TRUE or FALSE. Turns tracing on or off.</td>
</tr>
<tr>
<td>collection_name</td>
<td>The Oracle TRACE collection name to be used.</td>
</tr>
<tr>
<td>facility_name</td>
<td>The Oracle TRACE facility name to be used.</td>
</tr>
</tbody>
</table>
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 `OTRCSV.R.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

```sql
EXECUTE DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION
(8,12,TRUE,'NEWCOLL','oracled');
```
SET_ORACLE_TRACE_IN_SESSION Procedure
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
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_off</td>
<td>TRUE or FALSE: Turns tracing on or off.</td>
</tr>
<tr>
<td>collection_name</td>
<td>Oracle TRACE collection name to be used.</td>
</tr>
<tr>
<td>facility_name</td>
<td>Oracle TRACE facility name to be used.</td>
</tr>
</tbody>
</table>

Example

```sql
EXECUTE DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE (TRUE,'MYCOLL','oracle');
```
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 recreate 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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DROP_BY_CAT Procedure&quot; on page 37-1</td>
<td>Drops outlines that belong to a specified category.</td>
</tr>
<tr>
<td>&quot;DROP_COLLISION Procedure&quot; on page 37-3</td>
<td>Drops an outline with an ol$.hintcount value that does not match the number of hints for that outline in ol$hints.</td>
</tr>
<tr>
<td>&quot;DROP_EXTRAS Procedure&quot; on page 37-3</td>
<td>Cleans up after an import by dropping extra hint tuples not accounted for by hintcount.</td>
</tr>
<tr>
<td>&quot;DROP_UNREFD_HINTS Procedure&quot; on page 37-4</td>
<td>Drops hint tuples that have no corresponding outline in the OL$ table.</td>
</tr>
<tr>
<td>&quot;DROP_BY_CAT Procedure&quot; on page 37-2</td>
<td>Drops outlines that have never been applied in the compilation of a SQL statement.</td>
</tr>
<tr>
<td>&quot;UPDATE_BY_CAT Procedure&quot; on page 37-4</td>
<td>Changes the category of outlines in one category to a new category.</td>
</tr>
<tr>
<td>&quot;GENERATE_SIGNATURE Procedure&quot; on page 37-5</td>
<td>Generates a signature for the specified SQL text.</td>
</tr>
</tbody>
</table>

DROP_BY_CAT Procedure
This procedure drops outlines that belong to a specified category.

Syntax

DBMS_OUTLN.DROP_BY_CAT
Summary of DBMS_OUTLN Subprograms

Parameters

Table 37–2 DROP_BY_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>Category of outlines to drop.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure purges a category of outlines in a single call.

Example

This example drops all outlines in the DEFAULT category:

```sql
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

```sql
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

```sql
DBMS_OUTLN.DROP_EXTRAS;
```
**Usage Notes**

The OLS-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**

```sql
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**

```sql
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**

```sql
DBMS.OUTLN.UPDATE_BY_CAT (  
    oldcat VARCHAR2 DEFAULT 'DEFAULT',  
    newcat VARCHAR2 DEFAULT 'DEFAULT');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldcat</td>
<td>Current category to be changed.</td>
</tr>
<tr>
<td>newcat</td>
<td>Target category to change outline to.</td>
</tr>
</tbody>
</table>

**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:

```sql
DBMS_OUTLN.UPDATE_BY_CAT('DEFAULT', 'CAT1');
```

**GENERATE_SIGNATURE Procedure**

This procedure generates a signature for the specified SQL text.

**Syntax**

```sql
DBMS.OUTLN.GENERATE_SIGNATURE (  
    sqltxt IN VARCHAR2,  
    signature OUT RAW);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqltxt</td>
<td>The specified SQL.</td>
</tr>
<tr>
<td>signature</td>
<td>The signature to be generated.</td>
</tr>
</tbody>
</table>
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 38–1  DBMS_OUTLN_EDIT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CHANGE_JOIN_POS Procedure&quot; on page 38-2</td>
<td>Changes the join position for the hint identified by outline name and hint number to the position specified by newpos.</td>
</tr>
<tr>
<td>&quot;CREATE_EDIT_TABLES Procedure&quot; on page 38-2</td>
<td>Creates outline editing tables in calling a user’s schema.</td>
</tr>
<tr>
<td>&quot;DROP_EDIT_TABLES Procedure&quot; on page 38-3</td>
<td>Drops outline editing tables in calling the user's schema.</td>
</tr>
<tr>
<td>&quot;REFRESH_PRIVATE_OUTLINE Procedure&quot; on page 38-3</td>
<td>Refreshes the in-memory copy of the outline, synchronizing its data with the edits made to the outline hints.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_OUTLN_EDIT.CHANGE_JOIN_POS (name VARCHAR2, hintno NUMBER, newpos NUMBER);
```

Parameters

Table 38–2  CHANGE_JOIN_POS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the private outline to be modified.</td>
</tr>
<tr>
<td>hintno</td>
<td>Hint number to be modified.</td>
</tr>
<tr>
<td>newpos</td>
<td>New join position for the target hint.</td>
</tr>
</tbody>
</table>

CREATE_EDIT_TABLES Procedure

This procedure creates outline editing tables in calling a user’s schema.
Summary of DBMS_OUTLN_EDIT Subprograms

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the private outline to be refreshed.</td>
</tr>
</tbody>
</table>
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**
- Using **DBMS_OUTPUT**
- Summary of **DBMS_OUTPUT** Subprograms
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>
<thead>
<tr>
<th>Table 39–1 DBMS_OUTPUT Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
</tr>
<tr>
<td>ORU-10027:</td>
</tr>
<tr>
<td>ORU-10028:</td>
</tr>
</tbody>
</table>

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:

```sql
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:

```sql
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 non-zero. 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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ENABLE Procedure&quot; on page 39-3</td>
<td>Enables message output.</td>
</tr>
<tr>
<td>&quot;DISABLE Procedure&quot; on page 39-4</td>
<td>Disables message output.</td>
</tr>
<tr>
<td>&quot;PUT and PUT_LINE Procedures&quot; on page 39-4</td>
<td>PUT: Places a line in the buffer. PUT_LINE: Places partial line in buffer.</td>
</tr>
<tr>
<td>&quot;NEW_LINE Procedure&quot; on page 39-6</td>
<td>Terminates a line created with PUT.</td>
</tr>
<tr>
<td>&quot;GET_LINE and GET_LINES Procedures&quot; on page 39-6</td>
<td>Retrieves one line, or an array of lines, from buffer.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_OUTPUT.ENABLE (  
    buffer_size IN INTEGER DEFAULT 20000);  
```
DISABLE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size</td>
<td>Amount of information, in bytes, to buffer.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000;</td>
<td>Buffer overflow, limit of &lt;buffer_limit&gt; bytes.</td>
</tr>
<tr>
<td>ORU-10027:</td>
<td></td>
</tr>
</tbody>
</table>

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(enable,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_

---

39-4  Supplied PL/SQL Packages and Types Reference
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

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_OUTPUT.PUT</td>
<td>(item IN NUMBER);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.PUT</td>
<td>(item IN VARCHAR2);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.PUT</td>
<td>(item IN DATE);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.PUT_LINE</td>
<td>(item IN NUMBER);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.PUT_LINE</td>
<td>(item IN VARCHAR2);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.PUT_LINE</td>
<td>(item IN DATE);</td>
</tr>
<tr>
<td>DBMS_OUTPUT.NEW_LINE;</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Item to buffer.</td>
</tr>
</tbody>
</table>
NEW_LINE Procedure

Errors

Table 39–6 PUT and PUT_LINE Procedure Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000,</td>
<td>Buffer overflow, limit of (&lt;\text{buf_limit})) bytes.</td>
</tr>
<tr>
<td>ORU-10027:</td>
<td></td>
</tr>
<tr>
<td>ORA-20000,</td>
<td>Line length overflow, limit of 255 bytes per line.</td>
</tr>
<tr>
<td>ORU-10028:</td>
<td></td>
</tr>
</tbody>
</table>

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

```sql
DBMS_OUTPUT.NEW_LINE;
```

Errors

Table 39–7 NEW_LINE Procedure Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000,</td>
<td>Buffer overflow, limit of (&lt;\text{buf_limit})) bytes.</td>
</tr>
<tr>
<td>ORU-10027:</td>
<td></td>
</tr>
<tr>
<td>ORA-20000,</td>
<td>Line length overflow, limit of 255 bytes per line.</td>
</tr>
<tr>
<td>ORU-10028:</td>
<td></td>
</tr>
</tbody>
</table>

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**

```sql
DBMS_OUTPUT.GET_LINE (  
    line OUT VARCHAR2,  
    status OUT INTEGER);  
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>Returns a single line of buffered information, excluding a final newline character. The maximum length is 255 bytes.</td>
</tr>
<tr>
<td>status</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Syntax**

```sql
DBMS_OUTPUT.GET_LINES (  
    lines OUT CHARARR,  
    numlines IN OUT INTEGER);  
```

CHARARR is a table of VARCHAR2(255).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines</td>
<td>Returns an array of lines of buffered information. The maximum length of each line in the array is 255 bytes.</td>
</tr>
<tr>
<td>numlines</td>
<td>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.</td>
</tr>
</tbody>
</table>
Examples

The DBMS_OUTPUT package is commonly used to debug stored procedures and triggers, as shown in Example 1. This package can also be used to enable you to retrieve information about an object and format this output, as shown in Example 2 on page 39-9.

Example 1 This is an example of a function that 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:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>1500</td>
<td>500</td>
<td>20</td>
</tr>
<tr>
<td>1203</td>
<td>1000</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>1289</td>
<td>1000</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1347</td>
<td>1000</td>
<td>250</td>
<td>20</td>
</tr>
</tbody>
</table>

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** 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.

```sql
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* procedure, to uniquely identify statement. */
CREATE OR REPLACE PROCEDURE explain_out
    (statement_id IN VARCHAR2) AS

    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 indentation level.

DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) || line.operation || ' ' || line.options || ' ' || line.object_name);
END LOOP;
END;

See Also: Chapter 77, "UTL_FILE"
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 per partition is utilized.

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 8i.

This chapter discusses the following topics:

- Using DBMS_PCLXUTIL
- Limitations
- Summary of DBMS_PCLUTTL Subprograms
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.

   ```sql
   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.

   ```sql
   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
   ```

   For composite partitions, the procedure automatically builds local indices 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.

   ```sql
   ALTER INDEX <idx_name> local(...) usable;
   ```

   Rebuild only the desired (sub)partitions (that are marked unusable):

   ```sql
   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
   ```

   Rebuild all (sub)partitions using force_opt = TRUE:

   ```sql
   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 initialization 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 simply prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the snp*.trc trace files.

- The primary ramification of the above point is that you are expected to know how to tune Oracle (especially to set various storage parameters) in order to build large indexes. This package is not intended to assist in that tuning process.

Summary of DBMS_PCLUTTL Subprograms

DBMS_PCLXUTIL contains just one procedure: BUILD_PART_INDEX.

BUILD_PART_INDEX Procedure

Syntax

```sql
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);
```
BUILD_PART_INDEX Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procs_per_job</td>
<td>Number of parallel query slaves to be utilized per local index build (1 ≤ procs_per_job ≤ max_slaves).</td>
</tr>
<tr>
<td>tab_name</td>
<td>Name of the partitioned table (an exception is raised if the table does not exist or not partitioned).</td>
</tr>
<tr>
<td>idx_name</td>
<td>Name given to the local index (an exception is raised if a local index is not created on the table tab_name).</td>
</tr>
<tr>
<td>force_opt</td>
<td>If TRUE, then force rebuild of all partitioned indices; otherwise, rebuild only the partitions marked 'UNUSABLE'.</td>
</tr>
</tbody>
</table>

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
```
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 multi-threaded manner, 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>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23321:</td>
<td>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.</td>
</tr>
<tr>
<td>ORA-23322:</td>
<td>Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.</td>
</tr>
</tbody>
</table>

Summary of DBMS_PIPE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“CREATE_PIPE Function” on page 41-5</td>
<td>Explicitly creates a pipe (necessary for private pipes).</td>
</tr>
<tr>
<td>“PACK_MESSAGE Procedure” on page 41-7</td>
<td>Builds message in local buffer.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_PIPE.CREATE_PIPE (  
    pipename IN VARCHAR2,  
    maxpipesize IN INTEGER DEFAULT 8192,  
    private IN BOOLEAN DEFAULT TRUE)  
RETURN INTEGER;
```

Pragmas

```sql
pragma restrict_references(create_pipe,WNDS,RNDS);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>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.</td>
</tr>
<tr>
<td>maxpipesize</td>
<td>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.</td>
</tr>
<tr>
<td>private</td>
<td>Uses the default, TRUE, to create a private pipe. Public pipes can be implicitly created when you call SEND_MESSAGE.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 41–4 CREATE_PIPE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23322</td>
<td>Failure due to naming conflict.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 41–5 CREATE_PIPE Function Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error: Pipe with the same name already exists, and you are not allowed to use it.</td>
</tr>
</tbody>
</table>

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);
**SEND_MESSAGE Function**

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**

```sql
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 41–6  PACK_MESSAGE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Item to pack into the local message buffer.</td>
</tr>
</tbody>
</table>

---

**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**

```sql
DBMS_PIPE.SEND_MESSAGE (    pipename IN VARCHAR2,
    timeout IN INTEGER DEFAULT MAXWAIT,
    maxpipesize IN INTEGER DEFAULT 8192)
RETURN INTEGER;
```

---

**Pragmas**

```sql
pragma restrict_references(send_message,WNDS,RNDS);
```
## Parameters

**Table 41–7  SEND_MESSAGE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>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.</td>
</tr>
<tr>
<td>timeout</td>
<td>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).</td>
</tr>
<tr>
<td>maxpipesize</td>
<td>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.</td>
</tr>
</tbody>
</table>
RECEIVE_MESSAGE Function

Returns

Table 41–8  SEND_MESSAGE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>An interrupt occurred. If the pipe was implicitly created and is empty, then it is removed.</td>
</tr>
<tr>
<td>ORA–23322</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 41–9  SEND_Message Function Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.</td>
</tr>
</tbody>
</table>

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**

```sql
DBMS_PIPE.RECEIVE_MESSAGE (  
    pipename IN VARCHAR2,
    timeout IN INTEGER DEFAULT maxwait)
RETURN INTEGER;
```

**Pragmas**

```sql
pragma restrict_references(receive_message,WND,RND);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of the pipe on which you want to receive a message. Names beginning with <code>ORA$</code> are reserved for use by Oracle.</td>
</tr>
<tr>
<td>timeout</td>
<td>Time to wait for a message, in seconds. The default value is the constant <code>MAXWAIT</code>, which is defined as 86400000 (1000 days). A timeout of 0 allows you to read without blocking.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_PIPE.NEXT_ITEM_TYPE
    RETURN INTEGER;
```

Pragmas

```sql
pragma restrict_references(next_item_type,WNDS,RNDS);
```

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No more items</td>
</tr>
<tr>
<td>6</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE</td>
<td>(item OUT VARCHAR2);</td>
</tr>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE</td>
<td>(item OUT NCHAR);</td>
</tr>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE</td>
<td>(item OUT NUMBER);</td>
</tr>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE</td>
<td>(item OUT DATE);</td>
</tr>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE_RAW</td>
<td>(item OUT RAW);</td>
</tr>
<tr>
<td>DBMS_PIPE.UNPACK_MESSAGE_ROWID</td>
<td>(item OUT ROWID);</td>
</tr>
</tbody>
</table>

**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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Argument to receive the next unpacked item from the local message buffer.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_PIPE.REMOVE_PIPE (  
    pipename IN VARCHAR2)  
RETURN INTEGER;
```

**Pragmas**

```sql
pragma restrict_references(remove_pipe,WNDS,RNDS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of pipe that you want to remove.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_PIPE Subprograms

**Exceptions**

**Table 41–16 REMOVE_PIPE Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA–23322</td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td></td>
<td>If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA–23322, indicating insufficient privileges.</td>
</tr>
</tbody>
</table>

**Table 41–17 REMOVE_PIPE Function Exception**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_PIPE.PURGE (    pipename  IN  VARCHAR2);```

**Pragmas**

```sql
pragma restrict_references(purge,WNDS,RNDS);```
RESET_BUFFER Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>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.</td>
</tr>
</tbody>
</table>

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;

Pragma

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) THEN
        printf("\n%.s\n", msg_length, retval);
    ELSE
        printf("abnormal status, value is %d\n", status);
    END IF;
}

void sql_error()
{
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,
```
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;
BEGIN

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()
Summary of DBMS_PIPE Subprograms

```c
{
    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.UNPACKMESSAGE(: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';
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):

```sql
<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:

```sql
BEGIN :price := stock_request('YOURCOMPANY'); end;
```

The stored procedure, `dbms_stock_server`, which is called by the "stock price request server" above is:

```sql
CREATE OR REPLACE PACKAGE dbms_stock_server IS
    PROCEDURE get_request(symbol OUT VARCHAR2);
    PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;
```

```sql
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
            -- Handle error
        END;
    END;
END;
```
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;
The procedure called by the client is:

```sql
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"
Oracle8i 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.

This chapter discusses the following topics:

- Using `DBMS_PROFILER`
- Requirements
- Security
- Exceptions
- Error Codes
- Summary of `DBMS_PROFILER` Subprograms
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 42-8.

**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.
The profiling information is stored in database tables. This enables ad-hoc querying on the data: you can build customizable reports (summary reports, hottest lines, code coverage data, and so on). It also allows you to analyze the data.

**PROFTAB.SQL**

The PROFTAB.SQL script creates tables with the columns, datatypes, and definitions as shown in Table 42–1, Table 42–2, and Table 42–3.

**Table 42–1  Columns in Table PLSQL_PROFILER_RUNS**

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>number</td>
<td>Unique run identifier from plsql_profiler_runnumber</td>
</tr>
<tr>
<td>related_run</td>
<td>number</td>
<td>Runid of related run (for client/server correlation)</td>
</tr>
<tr>
<td>run_owner</td>
<td>varchar2(32)</td>
<td>User who started run</td>
</tr>
<tr>
<td>run_date</td>
<td>date</td>
<td>Start time of run</td>
</tr>
<tr>
<td>run_comment</td>
<td>varchar2(2047)</td>
<td>User provided comment for this run</td>
</tr>
<tr>
<td>run_total_time</td>
<td>number</td>
<td>Elapsed time for this run in nanoseconds</td>
</tr>
<tr>
<td>run_system_info</td>
<td>varchar2(2047)</td>
<td>Currently unused</td>
</tr>
<tr>
<td>run_comment1</td>
<td>varchar2(2047)</td>
<td>Additional comment</td>
</tr>
<tr>
<td>spare1</td>
<td>varchar2(256)</td>
<td>Unused</td>
</tr>
</tbody>
</table>

**Table 42–2  Columns in Table PLSQL_PROFILER_UNITS**

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>number</td>
<td>Primary key, references plsql_profiler_runs,</td>
</tr>
<tr>
<td>unit_number</td>
<td>number</td>
<td>Primary key, internally generated library unit #</td>
</tr>
<tr>
<td>unit_type</td>
<td>varchar2(32)</td>
<td>Library unit type</td>
</tr>
<tr>
<td>unit_owner</td>
<td>varchar2(32)</td>
<td>Library unit owner name</td>
</tr>
<tr>
<td>unit_name</td>
<td>varchar2(32)</td>
<td>Library unit name timestamp on library unit</td>
</tr>
</tbody>
</table>
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 42–4 shows the exceptions for DBMS_PROFILER.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version_mismatch</td>
<td>Corresponds to error_version.</td>
</tr>
<tr>
<td>profiler_error</td>
<td>Corresponds to either &quot;error_param&quot; or &quot;error_io&quot;.</td>
</tr>
</tbody>
</table>

**Error Codes**

A 0 return value from any function denotes successful completion; a non-zero return value denotes an error condition. The possible errors are listed below:

- ‘A subprogram was called with an incorrect parameter.’
  ```plsql```
  ```
  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.’
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.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;START_PROFILER Function&quot; on page 42-7</td>
<td>Starts profiler data collection in the user’s session.</td>
</tr>
<tr>
<td>&quot;STOP_PROFILER Function&quot; on page 42-8</td>
<td>Stops profiler data collection in the user’s session.</td>
</tr>
<tr>
<td>&quot;FLUSH_DATA Function&quot; on page 42-8</td>
<td>Flushes profiler data collected in the user’s session.</td>
</tr>
<tr>
<td>&quot;PAUSE_PROFILER Function&quot; on page 42-9</td>
<td>Pauses profiler data collection.</td>
</tr>
<tr>
<td>&quot;RESUME_PROFILER Function&quot; on page 42-9</td>
<td>Resumes profiler data collection.</td>
</tr>
<tr>
<td>&quot;GET_VERSION Procedure&quot; on page 42-9</td>
<td>Gets the version of this API.</td>
</tr>
<tr>
<td>&quot;INTERNAL_VERSION_CHECK Function&quot; on page 42-9</td>
<td>Verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.</td>
</tr>
</tbody>
</table>

**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
STOP_PROFILER Function

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 42–6  START_PROFILER Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_comment</td>
<td>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.</td>
</tr>
<tr>
<td>run_number</td>
<td>Stores the number of the run so you can store and later recall the run’s data.</td>
</tr>
<tr>
<td>comment1</td>
<td>Allows you to make interesting comments about the run.</td>
</tr>
</tbody>
</table>

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 pre-exist.
Summary of DBMS_PROFILER Subprograms

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version of DBMS_PROFILER.</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version of DBMS_PROFILER.</td>
</tr>
</tbody>
</table>

INTERNAL_VERSION_CHECK Function
This function verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

Note: Use the PROFTAB.SQL script to create the tables and other data structures required for persistently storing the profiler data.
INTERNAL_VERSION_CHECK Function

Syntax

```
DBMS_PROFILER.INTERNAL_VERSION_CHECK
RETURN BINARY_INTEGER;
```
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 prior to 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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;INITIALIZE Procedure&quot; on page 43-2</td>
<td>Initializes the package with a seed value.</td>
</tr>
<tr>
<td>&quot;SEED Procedure&quot; on page 43-3</td>
<td>Resets the seed.</td>
</tr>
<tr>
<td>&quot;RANDOM Function&quot; on page 43-3</td>
<td>Gets the random number.</td>
</tr>
<tr>
<td>&quot;TERMINATE Procedure&quot; on page 43-3</td>
<td>Closes the package.</td>
</tr>
</tbody>
</table>

INITIALIZE Procedure

To use the package, first call the initialize subprogram with the seed to use.

Syntax

```sql
DBMS_RANDOM.INITIALIZE (seed IN BINARY_INTEGER);
```

Note: Use a seed that is sufficiently large, more than 5 digits. A single digit might not return sufficiently random numbers. Also consider getting the seed from variable values such as the time.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>Seed number used to generate a random number.</td>
</tr>
</tbody>
</table>
**SEED Procedure**

This procedure resets the seed.

**Syntax**

```sql
DBMS_RANDOM.SEED (
    seed IN BINARY_INTEGER);
```

**Parameters**

**Table 43–3  INITIALIZE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>Seed number used to generate a random number.</td>
</tr>
</tbody>
</table>

**RANDOM Function**

This function gets the random number.

**Syntax**

```sql
DBMS_RANDOM.RANDOM
    RETURN BINARY_INTEGER;
```

**TERMINATE Procedure**

When you are finished with the package, call the TERMINATE procedure.

**Syntax**

```sql
DBMS_RANDOM.TERMINATE;
```
TERMINATE Procedure
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
Summary of DBMS_RECTIFIER_DIFF Subprograms

Table 44–1  DBMS_RECTIFIER_DIFF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DIFFERENCES Procedure&quot; on page 44-3</td>
<td>Determines the differences between two tables.</td>
</tr>
<tr>
<td>&quot;RECTIFY Procedure&quot; on page 44-6</td>
<td>Resolves the differences between two tables.</td>
</tr>
</tbody>
</table>
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**

```sql
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 44–2  DIFFERENCES Procedure Parameters** (Page 1 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname1</td>
<td>Name of the schema at reference_site.</td>
</tr>
<tr>
<td>oname1</td>
<td>Name of the table at reference_site.</td>
</tr>
<tr>
<td>reference_site</td>
<td>Name of the reference database site. The default, NULL, indicates the current site.</td>
</tr>
<tr>
<td>sname2</td>
<td>Name of the schema at comparison_site.</td>
</tr>
<tr>
<td>oname2</td>
<td>Name of the table at comparison_site.</td>
</tr>
<tr>
<td>comparison_site</td>
<td>Name of the comparison database site. The default, NULL, indicates the current site.</td>
</tr>
<tr>
<td>where_clause</td>
<td>Only rows satisfying this clause are selected for comparison. The default, NULL, indicates all rows are compared.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>array_columns</td>
<td>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.</td>
</tr>
<tr>
<td>missing_rows_sname</td>
<td>Name of the schema containing the tables with the missing rows.</td>
</tr>
<tr>
<td>missing_rows_oname1</td>
<td>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.</td>
</tr>
<tr>
<td>missing_rows_oname2</td>
<td>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.</td>
</tr>
<tr>
<td>missing_rows_site</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Exceptions

#### Table 44–3 DIFFERENCES Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nosuchsite</td>
<td>Database site could not be found.</td>
</tr>
<tr>
<td>badnumber</td>
<td>The commit_rows parameter is less than 1.</td>
</tr>
<tr>
<td>missingprimarykey</td>
<td>Column list must include primary key (or SET_COLUMNS equivalent).</td>
</tr>
<tr>
<td>badname</td>
<td>NULL or empty string for table or schema name.</td>
</tr>
<tr>
<td>cannotbenull</td>
<td>Parameter cannot be NULL.</td>
</tr>
<tr>
<td>notshapeequivalent</td>
<td>Tables being compared are not shape equivalent. Shape refers to</td>
</tr>
<tr>
<td></td>
<td>the number of columns, their column names, and the column datatypes.</td>
</tr>
<tr>
<td>unknowncolumn</td>
<td>Column does not exist.</td>
</tr>
<tr>
<td>unsupportedtype</td>
<td>Type not supported.</td>
</tr>
<tr>
<td>dbms_repcat.commfailure</td>
<td>Remote site is inaccessible.</td>
</tr>
<tr>
<td>dbms_repcat.missingobject</td>
<td>Table does not exist.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname1</td>
<td>Name of the schema at reference_site.</td>
</tr>
<tr>
<td>oname1</td>
<td>Name of the table at reference_site.</td>
</tr>
<tr>
<td>reference_site</td>
<td>Name of the reference database site. The default, NULL, indicates the current site.</td>
</tr>
<tr>
<td>sname2</td>
<td>Name of the schema at comparison_site.</td>
</tr>
<tr>
<td>oname2</td>
<td>Name of the table at comparison_site.</td>
</tr>
<tr>
<td>comparison_site</td>
<td>Name of the comparison database site. The default, NULL, indicates the current site.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>array_columns</td>
<td>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.</td>
</tr>
<tr>
<td>missing_rows_sname</td>
<td>Name of the schema containing the tables with the missing rows.</td>
</tr>
<tr>
<td>missing_rows_oname1</td>
<td>Name of the table at missing_rows_site at which the row is present, and the name of the site from which the row is absent.</td>
</tr>
<tr>
<td>missing_rows_oname2</td>
<td>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.</td>
</tr>
<tr>
<td>missing_rows_site</td>
<td>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.</td>
</tr>
<tr>
<td>commit_rows</td>
<td>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.</td>
</tr>
</tbody>
</table>
RECTIFY Procedure

Exceptions

Table 44–5  RECTIFY Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nosuchsite</td>
<td>Database site could not be found.</td>
</tr>
<tr>
<td>badnumber</td>
<td>The commit_rows parameter is less than 1.</td>
</tr>
<tr>
<td>badname NULL or empty string for table or schema name.</td>
<td></td>
</tr>
<tr>
<td>dbms_repcat.commfailure</td>
<td>Remote site is inaccessible.</td>
</tr>
<tr>
<td>dbms_repcat.missingobject</td>
<td>Table does not exist.</td>
</tr>
</tbody>
</table>
With DBMS_REDEFINITION, you can perform an online reorganization of tables. To achieve this online reorganization, 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 reorganized online are as follows:

- Tables with no primary keys cannot be reorganized online.
- Tables that have materialized views and materialized view logs defined on them cannot be reorganized online.
- Tables that are materialized view container tables and AQ tables cannot be reorganized online.
- The overflow table of an IOT table cannot be reorganized online.

**See Also:** Oracle9i Database Administrator’s Guide for more information.

This chapter discusses the following topics:

- Summary of DBMS_REDEFINITION Subprograms
Summary of DBMS_REDEFINITION Subprograms

Table 45–1  DBMS_REDEFINITION Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“CAN_REDEF_TABLE Procedure”</td>
<td>Determines if a given table can be reorganized online.</td>
</tr>
<tr>
<td>“START_REDEF_TABLE Procedure”</td>
<td>Initiates the reorganization process.</td>
</tr>
<tr>
<td>“FINISH_REDEF_TABLE Procedure”</td>
<td>Completes the reorganization process.</td>
</tr>
<tr>
<td>“SYNC_INTERIM_TABLE Procedure”</td>
<td>Keeps the interim table synchronized with the original table.</td>
</tr>
<tr>
<td>“ABORT_REDEF_TABLE Procedure”</td>
<td>Cleans up errors that occur during the reorganization process.</td>
</tr>
</tbody>
</table>

CAN_REDEF_TABLE Procedure

This procedure determines if a given table can be reorganized online. This is the first step of the online reorganization process. If the table is not a candidate for online redefinition, an error message is raised.

Syntax

```sql
DBMS_REDEFINITION.can_redef_table (  
    uname IN VARCHAR2,  
    tname IN VARCHAR2);
```

Exceptions

If the table is not a candidate for online reorganization, an error message is raised.

Parameters

Table 45–2  CAN_REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>The schema name of the table.</td>
</tr>
<tr>
<td>tname</td>
<td>The name of the table to be reorganized.</td>
</tr>
</tbody>
</table>
**START_REDEF_TABLE Procedure**

This procedure initiates the reorganization process. After verifying that the table can be reorganized online, you create an empty interim table (in the same schema as the table to be reorganized) with the desired attributes of the post-reorganization table.

**Syntax**

```sql
DBMS_REDEFINITION.start_redef_table (  
    uname IN VARCHAR2,  
    orig_table IN VARCHAR2,  
    int_table IN VARCHAR2,  
    col_mapping IN VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>The schema name of the tables.</td>
</tr>
<tr>
<td>orig_table</td>
<td>The name of the table to be reorganized.</td>
</tr>
<tr>
<td>int_table</td>
<td>The name of the interim table.</td>
</tr>
<tr>
<td>col_mapping</td>
<td>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 <code>SELECT</code> clause of a query.) If NULL, all the columns in the original table are selected and have the same name after reorganization.</td>
</tr>
</tbody>
</table>

**FINISH_REDEF_TABLE Procedure**

This procedure completes the reorganization 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 reorganized with the attributes and data of the interim table. The original table is locked briefly during this procedure.

**Syntax**

```sql
DBMS_REDEFINITION.finish_redef_table (  
    uname IN VARCHAR2,  
    orig_table IN VARCHAR2,  
    int_table IN VARCHAR2);
```
SYNC_INTERIM_TABLE Procedure

Parameters

<table>
<thead>
<tr>
<th>Table 45–4 FINISH_REDEF_TABLE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td>uname</td>
</tr>
<tr>
<td>orig_table</td>
</tr>
<tr>
<td>int_table</td>
</tr>
</tbody>
</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 reorganization. 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>
<thead>
<tr>
<th>Table 45–5 SYNC_INTERIM_TABLE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>uname</td>
</tr>
<tr>
<td>orig_table</td>
</tr>
<tr>
<td>int_table</td>
</tr>
</tbody>
</table>

ABORT_REDEF_TABLE Procedure

This procedure cleans up errors that occur during the reorganization process. This procedure can also be used to abort the reorganization 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>The schema name of the tables.</td>
</tr>
<tr>
<td>orig_table</td>
<td>The name of the table to be reorganized.</td>
</tr>
<tr>
<td>int_table</td>
<td>The name of the interim table.</td>
</tr>
</tbody>
</table>
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
### Summary of DBMS_REFRESH Subprograms

**Table 46–1   DBMS_REFRESH Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD Procedure&quot; on page 46-3</td>
<td>Adds materialized views to a refresh group.</td>
</tr>
<tr>
<td>&quot;CHANGE Procedure&quot; on page 46-4</td>
<td>Changes the refresh interval for a refresh group.</td>
</tr>
<tr>
<td>&quot;DESTROY Procedure&quot; on page 46-6</td>
<td>Removes all of the materialized views from a refresh group and deletes the refresh group.</td>
</tr>
<tr>
<td>&quot;MAKE Procedure&quot; on page 46-7</td>
<td>Specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.</td>
</tr>
<tr>
<td>&quot;REFRESH Procedure&quot; on page 46-10</td>
<td>Manually refreshes a refresh group.</td>
</tr>
<tr>
<td>&quot;SUBTRACT Procedure&quot; on page 46-10</td>
<td>Removes materialized views from a refresh group.</td>
</tr>
</tbody>
</table>
ADD Procedure

This procedure adds materialized views to a refresh group.

**See Also:** Oracle9i Replication for more information

Syntax

```sql
DBMS_REFRESH.ADD (  
    name IN VARCHAR2,  
    { list IN VARCHAR2,  
    | tab IN DBMSUTILITY.UNCL_ARRAY, }  
    lax IN BOOLEAN := false);
```

**Note:** This procedure is overloaded. The list and tab parameters are mutually exclusive.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group to which you want to add members.</td>
</tr>
<tr>
<td>list</td>
<td>Comma-separated list of materialized views that you want to add to the refresh group. (Synonyms are not supported.)</td>
</tr>
<tr>
<td>tab</td>
<td>Instead of a comma-separated 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.</td>
</tr>
<tr>
<td>lax</td>
<td>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.</td>
</tr>
</tbody>
</table>
CHANGE Procedure

This procedure changes the refresh interval for a refresh group.

**See Also:** Oracle9i Replication for more information about refresh groups

**Syntax**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group for which you want to alter the refresh interval.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date that you want a refresh to occur. By default, this date remains unchanged.</td>
</tr>
<tr>
<td>interval</td>
<td>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.</td>
</tr>
<tr>
<td>implicit_destroy</td>
<td>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.</td>
</tr>
<tr>
<td>parallelism</td>
<td></td>
</tr>
<tr>
<td>heap_size</td>
<td></td>
</tr>
</tbody>
</table>
Summary of DBMS_REFRESH Subprograms

### Table 46–3 CHANGE Procedures Parameters (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rollback_seg</td>
<td>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.</td>
</tr>
<tr>
<td>push_deferred_rpc</td>
<td>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.</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>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.</td>
</tr>
<tr>
<td>purge_option</td>
<td>If you are using the parallel propagation mechanism (that is, parallelism is set to 1 or greater), then:</td>
</tr>
<tr>
<td></td>
<td>0 = do not purge</td>
</tr>
<tr>
<td></td>
<td>1 = lazy (default)</td>
</tr>
<tr>
<td></td>
<td>2 = aggressive</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>parallelism</td>
<td>0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>n &gt; 1 specifies parallel propagation with n parallel processes.</td>
</tr>
<tr>
<td></td>
<td>1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not set this parameter unless directed to do so by Oracle Support Services.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REFRESH.DESTROY (
    name    IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group that you want to destroy.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Unique name used to identify the refresh group. Refresh groups must follow the same naming conventions as tables.</td>
</tr>
<tr>
<td>list</td>
<td>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 current database.</td>
</tr>
<tr>
<td>tab</td>
<td>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.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date that you want a refresh to occur.</td>
</tr>
<tr>
<td>interval</td>
<td>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, &quot;MONDAY&quot;) 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.</td>
</tr>
<tr>
<td>implicit_destroy</td>
<td>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.</td>
</tr>
<tr>
<td>lax</td>
<td>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.</td>
</tr>
<tr>
<td>job</td>
<td>Needed by the Import utility. Use the default value, 0.</td>
</tr>
<tr>
<td>rollback_seg</td>
<td>Name of the rollback segment to use while refreshing materialized views. The default, NULL, uses the default rollback segment.</td>
</tr>
</tbody>
</table>
### Table 46–5 MAKE Procedure Parameters  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push_deferred_rpc</td>
<td>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.</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>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.</td>
</tr>
<tr>
<td>purge_option</td>
<td>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.</td>
</tr>
<tr>
<td>parallelism</td>
<td>0 specifies serial propagation. $n &gt; 1$ specifies parallel propagation with $n$ parallel processes. 1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

```sql
DBMS_REFRESH.REFRESH (name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Name of the refresh group that you want to refresh manually.</td>
</tr>
</tbody>
</table>

SUBTRACT Procedure

This procedure removes materialized views from a refresh group.

**See Also:** Oracle9i Replication for more information about refresh groups

**Syntax**

```sql
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 46–7  SUBTRACT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group from which you want to remove members.</td>
</tr>
<tr>
<td>list</td>
<td>Comma-separated 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.</td>
</tr>
<tr>
<td>tab</td>
<td>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.</td>
</tr>
<tr>
<td>lax</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 47–1 lists the parameters and the enumerated constants.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>TABLE_OBJECT, INDEX_OBJECT, CLUSTER_OBJECT</td>
</tr>
<tr>
<td>action</td>
<td>CREATE_ACTION, DROP_ACTION, PURGE_ACTION</td>
</tr>
<tr>
<td>table_type</td>
<td>REPAIR_TABLE, ORPHAN_TABLE</td>
</tr>
<tr>
<td>flags</td>
<td>SKIP_FLAG, NOSKIP_FLAG</td>
</tr>
</tbody>
</table>

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 47–2 DBMS_REPAIR Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>942</td>
<td>Reported by DBMS_REPAIR.ADMIN_TABLES during a DROP_ACTION when the specified table doesn’t exist.</td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>Reported by DBMS_REPAIR.CREATE_ACTION when the specified table already exists.</td>
<td></td>
</tr>
<tr>
<td>Exception</td>
<td>Description</td>
<td>Action</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>24120</td>
<td>An invalid parameter was passed to the specified DBMS_REPAIR procedure.</td>
<td>Specify a valid parameter value or use the parameter’s default.</td>
</tr>
<tr>
<td>24122</td>
<td>An incorrect block range was specified.</td>
<td>Specify correct values for the BLOCK_START and BLOCK_END parameters.</td>
</tr>
<tr>
<td>24123</td>
<td>An attempt was made to use the specified feature, but the feature is not yet implemented.</td>
<td>Do not attempt to use the feature.</td>
</tr>
<tr>
<td>24124</td>
<td>An invalid ACTION parameter was specified.</td>
<td>Specify CREATE_ACTION, PURGE_ACTION or DROP_ACTION for the ACTION parameter.</td>
</tr>
<tr>
<td>24125</td>
<td>An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since DBMS_REPAIR.CHECK_OBJECT was run.</td>
<td>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.</td>
</tr>
<tr>
<td>24127</td>
<td>TABLESPACE parameter specified with an ACTION other than CREATE_ACTION.</td>
<td>Do not specify TABLESPACE when performing actions other than CREATE_ACTION.</td>
</tr>
<tr>
<td>24128</td>
<td>A partition name was specified for an object that is not partitioned.</td>
<td>Specify a partition name only if the object is partitioned.</td>
</tr>
<tr>
<td>24129</td>
<td>An attempt was made to pass a table name parameter without the specified prefix.</td>
<td>Pass a valid table name parameter.</td>
</tr>
<tr>
<td>24130</td>
<td>An attempt was made to specify a repair or orphan table that does not exist.</td>
<td>Specify a valid table name parameter.</td>
</tr>
<tr>
<td>24131</td>
<td>An attempt was made to specify a repair or orphan table that does not have a correct definition.</td>
<td>Specify a table name that refers to a properly created table.</td>
</tr>
<tr>
<td>24132</td>
<td>An attempt was made to specify a table name is greater than 30 characters long.</td>
<td>Specify a valid table name parameter.</td>
</tr>
</tbody>
</table>
Summary of DBMS_REPAIR Subprograms

Table 47–3  DBMS_REPAIR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADMIN_TABLES Procedure&quot; on page 47-4</td>
<td>Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions.</td>
</tr>
<tr>
<td>&quot;CHECK_OBJECT Procedure&quot; on page 47-5</td>
<td>Detects and reports corruptions in a table or index.</td>
</tr>
<tr>
<td>&quot;DUMP_ORPHAN_KEYS Procedure&quot; on page 47-7</td>
<td>Reports on index entries that point to rows in corrupt data blocks.</td>
</tr>
<tr>
<td>&quot;FIX_CORRUPT_BLOCKS Procedure&quot; on page 47-8</td>
<td>Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT.</td>
</tr>
<tr>
<td>&quot;REBUILD_FREELISTS Procedure&quot; on page 47-9</td>
<td>Rebuilds an object’s freelists.</td>
</tr>
<tr>
<td>&quot;SKIP_CORRUPT_BLOCKS Procedure&quot; on page 47-10</td>
<td>Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered.</td>
</tr>
<tr>
<td>&quot;SEGMENT_FIX_STATUS Procedure&quot; on page 47-11</td>
<td>Fixes the corrupted state of a bitmap entry.</td>
</tr>
</tbody>
</table>

ADMIN_TABLES Procedure

This procedure provides administrative functions for the DBMS_REPAIR package repair and orphan key tables.

Syntax

```sql
DBMS_REPAIR.ADMIN_TABLES ( 
  table_name IN VARCHAR2, 
  table_type IN BINARY_INTEGER, 
  action IN BINARY_INTEGER, 
  tablespace IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 47–4  ADMIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>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_.</td>
</tr>
<tr>
<td>table_type</td>
<td>Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE. See &quot;Enumeration Types&quot; on page 47-2.</td>
</tr>
<tr>
<td>action</td>
<td>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_&lt;table_name&gt; 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 &quot;Enumeration Types&quot; on page 47-2.</td>
</tr>
<tr>
<td>tablespace</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 ( 
```
CHECK_OBJECT Procedure

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name of the object to be checked.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table or index to be checked.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be checked.</td>
</tr>
<tr>
<td></td>
<td>If this is a partitioned object, and if <code>partition_name</code> 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.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See &quot;Enumeration Types&quot; on page 47-2.</td>
</tr>
<tr>
<td>repair_table_name</td>
<td>Name of the repair table to be populated. The table must exist in the SYS schema. Use the <code>admin_tables</code> procedure to create a repair table. The default name is REPAIR_TABLE.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number: Used when specifying a block range.</td>
</tr>
<tr>
<td>block_start</td>
<td>First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.</td>
</tr>
<tr>
<td>block_end</td>
<td>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 <code>block_start</code> or <code>block_end</code> is specified, then the other defaults to the first or last block in the file respectively.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Object name.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be processed.</td>
</tr>
<tr>
<td>object_type</td>
<td></td>
</tr>
<tr>
<td>repair_table_name</td>
<td></td>
</tr>
<tr>
<td>orphan_table_name</td>
<td></td>
</tr>
<tr>
<td>flags</td>
<td></td>
</tr>
<tr>
<td>key_count</td>
<td></td>
</tr>
</tbody>
</table>

Table 47–6  DUMP_ORPHAN_KEYS Procedure Parameters

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.
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

```sql
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 47–7  FIX_CORRUPT_BLOCKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object with corrupt blocks to be fixed.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be processed.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT.</td>
</tr>
<tr>
<td>repair_table_name</td>
<td>Name of the repair table with the repair directives.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>fix_count</td>
<td>Number of blocks fixed.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPAIR.REBUILD_FREELISTS (  
    schema_name   IN VARCHAR2,  
    partition_name IN VARCHAR2 DEFAULT NULL,  
    object_type   IN BINARY_INTEGER DEFAULT TABLE_OBJECT);  
```
**SKIP_CORRUPT_BLOCKS Procedure**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object whose freelists are to be rebuilt.</td>
</tr>
<tr>
<td>partition_name</td>
<td>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.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See “Enumeration Types” on page 47-2.</td>
</tr>
</tbody>
</table>

**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**

```sql
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 47–9  SKIP_CORRUPT_BLOCKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name of the object to be processed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be processed.</td>
</tr>
<tr>
<td>(optional)</td>
<td>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.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or CLUSTER_OBJECT.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Enumeration Types&quot; on page 47-2.</td>
</tr>
<tr>
<td>flags</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Enumeration Types&quot; on page 47-2.</td>
</tr>
</tbody>
</table>

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

```sql
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,)
```
SEGMENT_FIX_STATUS Procedure

Parameters

Table 47–10  SEGMENT_FIX_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_owner</td>
<td>Schema name of the segment.</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Optional. Name of an individual partition. NULL for nonpartitioned objects. Default is NULL.</td>
</tr>
<tr>
<td>segment_type</td>
<td>Optional Type of the segment (for example, TABLE or INDEX). Default is NULL.</td>
</tr>
<tr>
<td>file_number</td>
<td>(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.</td>
</tr>
<tr>
<td>block_number</td>
<td>(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.</td>
</tr>
<tr>
<td>status_value</td>
<td>(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.</td>
</tr>
</tbody>
</table>

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);
DBMS_REPCAT provides routines to administer and update the replication catalog and environment.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT Subprograms
### Summary of DBMS_REPCAT Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_GROUPED_COLUMN Procedure&quot; on page 48-8</td>
<td>Adds members to an existing column group.</td>
</tr>
<tr>
<td>&quot;ADD_MASTER_DATABASE Procedure&quot; on page 48-9</td>
<td>Adds another master site to your replication environment.</td>
</tr>
<tr>
<td>&quot;ADD_NEW_MASTERS Procedure&quot; on page 48-11</td>
<td>Adds the master sites in the <code>DBA_REPSITES_NEW</code> data dictionary view to the replication catalog at all available master sites.</td>
</tr>
<tr>
<td>&quot;ADD_PRIORITY_datatype Procedure&quot; on page 48-17</td>
<td>Adds a member to a priority group.</td>
</tr>
<tr>
<td>&quot;ADD_SITE_PRIORITY_SITE Procedure&quot; on page 48-19</td>
<td>Adds a new site to a site priority group.</td>
</tr>
<tr>
<td>&quot;ADD_conflictype_RESOLUTION Procedure&quot; on page 48-20</td>
<td>Designates a method for resolving an update, delete, or uniqueness conflict.</td>
</tr>
<tr>
<td>&quot;ALTER_CATCHUP_PARAMETERS Procedure&quot; on page 48-26</td>
<td>Alters the values for parameters stored in the <code>DBA_REPEXTENSIONS</code> data dictionary view.</td>
</tr>
<tr>
<td>&quot;ALTER_MASTER_PROPAGATION Procedure&quot; on page 48-28</td>
<td>Alters the propagation method for a specified replication group at a specified master site.</td>
</tr>
<tr>
<td>&quot;ALTER_MASTER_REPOBJECT Procedure&quot; on page 48-29</td>
<td>Alters an object in your replication environment.</td>
</tr>
<tr>
<td>&quot;ALTER_MVIEW_PROPAGATION Procedure&quot; on page 48-32</td>
<td>Alters the propagation method for a specified replication group at the current materialized view site.</td>
</tr>
<tr>
<td>&quot;ALTER_PRIORITY Procedure&quot; on page 48-34</td>
<td>Alters the priority level associated with a specified priority group member.</td>
</tr>
<tr>
<td>&quot;ALTER_PRIORITY_datatype Procedure&quot; on page 48-35</td>
<td>Alters the value of a member in a priority group.</td>
</tr>
<tr>
<td>&quot;ALTER_SITE_PRIORITY Procedure&quot; on page 48-37</td>
<td>Alters the priority level associated with a specified site.</td>
</tr>
<tr>
<td>&quot;ALTER_SITE_PRIORITY_SITE Procedure&quot; on page 48-39</td>
<td>Alters the site associated with a specified priority level.</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;CANCEL_STATISTICS Procedure&quot; on page 48-40</td>
<td>Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_COLUMN_GROUP Procedure&quot; on page 48-41</td>
<td>Updates the comment field in the ALL_REPCOLUMN_GROUP view for a column group.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_conflicttype_RESOLUTION Procedure&quot; on page 48-48</td>
<td>Updates the SCHEMA_COMMENT field in the ALL_REPGROUP view for a materialized view site.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Procedures&quot; on page 48-43</td>
<td>Updates the comment field in the ALL_REPPRIORITY_GROUP view for a (site) priority group.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_REPGROUP Procedure&quot; on page 48-44</td>
<td>Updates the comment field in the ALL_REPGROUP view for a master group.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_REPOBJECT Procedure&quot; on page 48-45</td>
<td>Updates the comment field in the ALL_REPOBJECT view for a replicated object.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_REPSITES Procedure&quot; on page 48-46</td>
<td>Updates the comment field in the ALL_REPSITE view for a replicated site.</td>
</tr>
<tr>
<td>&quot;COMMENT_ON_conflicttype_RESOLUTION Procedure&quot; on page 48-48</td>
<td>Updates the comment field in the ALL_REPRISELATION view for a conflict resolution routine.</td>
</tr>
<tr>
<td>&quot;COMPARE_OLD_VALUES Procedure&quot; on page 48-50</td>
<td>Specifies whether to compare old column values at each master site for each nonkey column of a replicated table for updates and deletes.</td>
</tr>
<tr>
<td>&quot;CREATE_MASTER_REPGROUP Procedure&quot; on page 48-52</td>
<td>Creates a new, empty, quiesced master group.</td>
</tr>
<tr>
<td>&quot;CREATE_MASTER_REPOBJECT Procedure&quot; on page 48-53</td>
<td>Specifies that an object is a replicated object.</td>
</tr>
<tr>
<td>&quot;CREATE_MVIEW_REPGROUP Procedure&quot; on page 48-57</td>
<td>Creates a new, empty materialized view group in your local database.</td>
</tr>
<tr>
<td>&quot;CREATE_MVIEW_REPOBJECT Procedure&quot; on page 48-58</td>
<td>Adds a replicated object to a materialized view group.</td>
</tr>
<tr>
<td>&quot;DEFINE_COLUMN_GROUP Procedure&quot; on page 48-61</td>
<td>Creates an empty column group.</td>
</tr>
<tr>
<td>&quot;DEFINE_PRIORITY_GROUP Procedure&quot; on page 48-62</td>
<td>Creates a new priority group for a master group.</td>
</tr>
</tbody>
</table>
### Table 48–1 DBMS_REPCAT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DEFINE_SITE_PRIORITY Procedure&quot; on page 48-64</td>
<td>Creates a new site priority group for a master group.</td>
</tr>
<tr>
<td>&quot;DO_DEFERRED_REPCAT_ADMIN Procedure&quot; on page 48-65</td>
<td>Executes the local outstanding deferred administrative procedures for the specified master group at the current master site, or for all master sites.</td>
</tr>
<tr>
<td>&quot;DROP_COLUMN_GROUP Procedure&quot; on page 48-66</td>
<td>Drops a column group.</td>
</tr>
<tr>
<td>&quot;DROP_GROUPED_COLUMN Procedure&quot; on page 48-67</td>
<td>Removes members from a column group.</td>
</tr>
<tr>
<td>&quot;DROP_MASTER_REPGROUP Procedure&quot; on page 48-68</td>
<td>Drops a master group from your current site.</td>
</tr>
<tr>
<td>&quot;DROP_MASTER_REPOBJECT Procedure&quot; on page 48-69</td>
<td>Drops a replicated object from a master group.</td>
</tr>
<tr>
<td>&quot;DROP_PRIORITY Procedure&quot; on page 48-73</td>
<td>Drops a replicated object from a master group.</td>
</tr>
<tr>
<td>&quot;DROP_MVIEW_REPGROUP Procedure&quot; on page 48-71</td>
<td>Drops a materialized view site from your replication environment.</td>
</tr>
<tr>
<td>&quot;DROP_MVIEW_REPOBJECT Procedure&quot; on page 48-72</td>
<td>Drops a replicated object from a materialized view site.</td>
</tr>
<tr>
<td>&quot;DROP_PRIORITY Procedure&quot; on page 48-73</td>
<td>Drops a member of a priority group by priority level.</td>
</tr>
<tr>
<td>&quot;DROP_PRIORITY_GROUP Procedure&quot; on page 48-74</td>
<td>Drops a priority group for a specified master group.</td>
</tr>
<tr>
<td>&quot;DROP_PRIORITY_datatype Procedure&quot; on page 48-75</td>
<td>Drops a member of a priority group by value.</td>
</tr>
<tr>
<td>&quot;DROP_SITE_PRIORITY Procedure&quot; on page 48-76</td>
<td>Drops a site priority group for a specified master group.</td>
</tr>
<tr>
<td>&quot;DROP_SITE_PRIORITY_SITE Procedure&quot; on page 48-77</td>
<td>Drops a specified site, by name, from a site priority group.</td>
</tr>
<tr>
<td>&quot;DROP_conflicttype_RESOLUTION Procedure&quot; on page 48-78</td>
<td>Drops an update, delete, or uniqueness conflict resolution method.</td>
</tr>
<tr>
<td>&quot;EXECUTE_DDL Procedure&quot; on page 48-80</td>
<td>Supplies DDL that you want to have executed at each master site.</td>
</tr>
</tbody>
</table>
### Table 48–1  DBMS_REPCAT Package Subprograms  

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GENERATE_MVIEW_SUPPORT Procedure&quot; on page 48-82</td>
<td>Activates triggers and generate packages needed to support the replication of updatable materialized views or procedural replication.</td>
</tr>
<tr>
<td>&quot;GENERATE_REPLICATION_SUPPORT Procedure&quot; on page 48-83</td>
<td>Generates the triggers, packages, and procedures needed to support replication for a specified object.</td>
</tr>
<tr>
<td>&quot;MAKE_COLUMN_GROUP Procedure&quot; on page 48-85</td>
<td>Creates a new column group with one or more members.</td>
</tr>
<tr>
<td>&quot;PREPARE_INSTANTIATED_MASTER Procedure&quot; on page 48-87</td>
<td>Changes the global name of the database you are adding to a master group.</td>
</tr>
<tr>
<td>&quot;PURGE_MASTER_LOG Procedure&quot; on page 48-88</td>
<td>Removes local messages in the DBA_REPCATLOG associated with a specified identification number, source, or master group.</td>
</tr>
<tr>
<td>&quot;PURGE_STATISTICS Procedure&quot; on page 89</td>
<td>Removes information from the ALL_REPSOLUTION_STATISTICS view.</td>
</tr>
<tr>
<td>&quot;REFRESH_MVIEW_REPGROUP Procedure&quot; on page 48-90</td>
<td>Refreshes a materialized view group with the most recent data from its associated master site or master materialized view site.</td>
</tr>
<tr>
<td>&quot;REGISTER_MVIEW_REPGROUP Procedure&quot; on page 48-92</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;REGISTER_STATISTICS Procedure&quot; on page 48-94</td>
<td>Collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.</td>
</tr>
<tr>
<td>&quot;RELOCATE_MASTERDEF Procedure&quot; on page 95</td>
<td>Changes your master definition site to another master site in your replication environment.</td>
</tr>
<tr>
<td>&quot;REMOVE_MASTER_DATABASES Procedure&quot; on page 48-97</td>
<td>Removes one or more master databases from a replication environment.</td>
</tr>
<tr>
<td>&quot;RENAME_SHADOW_COLUMN_GROUP Procedure&quot; on page 48-98</td>
<td>Renames the shadow column group of a replicated table to make it a named column group.</td>
</tr>
<tr>
<td>Table 48–1</td>
<td>DBMS_REPCAT Package Subprograms</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>&quot;REPCAT_IMPORT_CHECK Procedure&quot; on page 48-99</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;RESUME_MASTER_ACTIVITY Procedure&quot; on page 48-100</td>
<td>Resumes normal replication activity after quiescing a replication environment.</td>
</tr>
<tr>
<td>&quot;RESUME_PROPAGATION_TO_MDEF Procedure&quot; on page 48-101</td>
<td>Indicates that export is effectively finished and propagation for both extended and unaffected replication groups existing at master sites can be enabled.</td>
</tr>
<tr>
<td>&quot;SEND_OLD_VALUES Procedure&quot; on page 48-102</td>
<td>Specifies whether to send old column values for each nonkey column of a replicated table for updates and deletes.</td>
</tr>
<tr>
<td>&quot;SET_COLUMNS Procedure&quot; on page 48-105</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;SPECIFY_NEW_MASTERS Procedure&quot; on page 48-107</td>
<td>Specifies the master sites you intend to add to an existing replication group without quiescing the group.</td>
</tr>
<tr>
<td>&quot;SUSPEND_MASTER_ACTIVITY Procedure&quot; on page 48-109</td>
<td>Suspends replication activity for a master group.</td>
</tr>
<tr>
<td>&quot;SWITCH_MVIEW_MASTER Procedure&quot; on page 48-110</td>
<td>Changes the master site of a materialized view group to another master site.</td>
</tr>
<tr>
<td>&quot;UNDO_ADD_NEW_MASTERS_REQUEST Procedure&quot; on page 48-111</td>
<td>Undoes all of the changes made by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures for a specified extension_id.</td>
</tr>
<tr>
<td>&quot;UNREGISTER_MVIEW_REPGROUP Procedure&quot; on page 48-114</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;VALIDATE Function&quot; on page 48-114</td>
<td>Validates the correctness of key conditions of a multimaster replication environment.</td>
</tr>
</tbody>
</table>
Summary of DBMS_REPCAT Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;WAIT_MASTER_LOG Procedure&quot; on page 48-118</td>
<td>Determines whether changes that were asynchronously propagated to a master site have been applied.</td>
</tr>
</tbody>
</table>
ADD_GROUPED_COLUMN Procedure

This procedure adds members to an existing column group. You must call this procedure from the master definition site.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table with which the column group is associated.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group to which you are adding members.</td>
</tr>
<tr>
<td>list_of_column_names</td>
<td>Names of the columns that you are adding to the designated column group.</td>
</tr>
</tbody>
</table>

This can either be a comma-separated 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_OIDS 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.
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

```sql
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);
```

Table 48–3 ADD_GROUPED_COLUMN Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>missinggroup</td>
<td>Specified column group does not exist.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>Specified column does not exist in the specified table.</td>
</tr>
<tr>
<td>duplicatecolumn</td>
<td>Specified column is already a member of another column group.</td>
</tr>
<tr>
<td>missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication group to which the specified table belongs is not quiesced.</td>
</tr>
</tbody>
</table>
ADD_MASTER_DATABASE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group being replicated. This replication group must already exist at the master definition site.</td>
</tr>
<tr>
<td>master</td>
<td>Fully qualified database name of the new master database.</td>
</tr>
<tr>
<td>use_existing_objects</td>
<td>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.</td>
</tr>
<tr>
<td>copy_rows</td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>This comment is added to the MASTER_COMMENT field of the DBA_REPSITES view.</td>
</tr>
<tr>
<td>propagation_mode</td>
<td>Method of forwarding changes to and receiving changes from new master database. Accepted values are synchronous and asynchronous.</td>
</tr>
<tr>
<td>fname</td>
<td>This parameter is for internal use only.</td>
</tr>
</tbody>
</table>

Note: Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication has not been suspended for the master group.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group does not exist at the specified database site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>New master is not accessible.</td>
</tr>
<tr>
<td>typefailure</td>
<td>An incorrect propagation mode was specified.</td>
</tr>
<tr>
<td>notcompat</td>
<td>Compatibility mode must be 7.3.0.0 or greater.</td>
</tr>
<tr>
<td>duplrepgroup</td>
<td>Master site already exists.</td>
</tr>
</tbody>
</table>
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 48-107
ADD_NEW_MASTERS Procedure

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>export_required</td>
<td>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.</td>
</tr>
<tr>
<td>available_master_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>available_master_table</td>
<td>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.</td>
</tr>
<tr>
<td>masterdef_flashback_scn</td>
<td>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.</td>
</tr>
<tr>
<td>extension_id</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 48–6 ADD_NEW_MASTERS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>break_trans_to_masterdef</td>
<td>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.</td>
</tr>
<tr>
<td>break_trans_to_new_masters</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 48–6  ADD_NEW_MASTERS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage_for_catchup_mdef</td>
<td>This parameter is meaningful only if <code>export_required</code> and <code>break_trans_to_masterdef</code> are both set to <code>true</code>. 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.</td>
</tr>
<tr>
<td>cycle_seconds_mdef</td>
<td>This parameter is meaningful when <code>percentage_for_catchup_mdef</code> 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.</td>
</tr>
<tr>
<td>percentage_for_catchup_new</td>
<td>This parameter is meaningful only if <code>break_trans_to_new_masters</code> is set to <code>true</code>. 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.</td>
</tr>
<tr>
<td>cycle_seconds_new</td>
<td>This parameter is meaningful when <code>percentage_for_catchup_new</code> 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.</td>
</tr>
</tbody>
</table>
Exceptions

Table 48–7  ADD_NEW_MASTERS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>typefailure</td>
<td>The parameter value specified for one of the parameters is not appropriate.</td>
</tr>
<tr>
<td>novalidextreq</td>
<td>No valid extension request. The extension_id is not valid.</td>
</tr>
<tr>
<td>nonewsites</td>
<td>No new master sites to be added for the specified extension request.</td>
</tr>
<tr>
<td>notanewsite</td>
<td>Not a new site for extension request. A site was specified that was not specified when you ran the SPECIFY_NEW_MASTERS procedure.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>

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 }
ADD_PRIORITY_datatype Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group for which you are creating a priority group.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group.</td>
</tr>
<tr>
<td>value</td>
<td>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.</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of this value. The higher the number, the higher the priority.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>duplicatevalue</td>
<td>Specified value already exists in the priority group.</td>
</tr>
<tr>
<td>duplicatepriority</td>
<td>Specified priority already exists in the priority group.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>Specified priority group does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified value has the incorrect datatype for the priority group.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (  
gname IN VARCHAR2,  
name IN VARCHAR2,  
site IN VARCHAR2,  
priority IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group for which you are adding a site to a group.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group to which you are adding a member.</td>
</tr>
<tr>
<td>site</td>
<td>Global database name of the site that you are adding.</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of the site that you are adding. A higher number indicates a higher priority level.</td>
</tr>
</tbody>
</table>
ADD_conflicttype_RESOLUTION Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingpriority</td>
<td>Specified site priority group does not exist.</td>
</tr>
<tr>
<td>duplicatepriority</td>
<td>Specified priority level already exists for another site in the group.</td>
</tr>
<tr>
<td>duplicatevalue</td>
<td>Specified site already exists in the site priority group.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Conflict Type</th>
<th>Procedure Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>ADD_UPDATE_RESOLUTION</td>
</tr>
<tr>
<td>uniqueness</td>
<td>ADD_UNIQUE_RESOLUTION</td>
</tr>
<tr>
<td>delete</td>
<td>ADD_DELETE_RESOLUTION</td>
</tr>
</tbody>
</table>

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
Summary of DBMS_REPCAT Subprograms

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 := NULL,  
  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);
```
ADD_conflictype_RESOLUTION Procedure

Parameters

Table 48–13  ADD_conflictype_RESOLUTION Procedure Parameters  (Page 1 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema containing the table to be replicated.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table to which you are adding a conflict resolution routine. The table can be the storage table of a nested table.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group to which you are adding a conflict resolution routine. Column groups are required for update conflict resolution routines only.</td>
</tr>
<tr>
<td>constraint_name</td>
<td>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.</td>
</tr>
<tr>
<td>sequence_no</td>
<td>Order in which the designated conflict resolution methods should be applied.</td>
</tr>
<tr>
<td>method</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 you 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-separated list of column names, or a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMS_UTIL.LNAME_ARRAY. Use DBMS_UTIL.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-separated 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 48-25 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>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist as a table in the specified schema using row-level replication.</td>
</tr>
<tr>
<td>missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>Column that you specified as part of the <code>parameter_column_name</code> parameter does not exist.</td>
</tr>
<tr>
<td>missinggroup</td>
<td>Specified column group does not exist.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>The priority group that you specified does not exist for the table.</td>
</tr>
<tr>
<td>invalidmethod</td>
<td>Resolution method that you specified is not recognized.</td>
</tr>
<tr>
<td>invalidparameter</td>
<td>Number of columns that you specified for the <code>parameter_column_name</code> parameter is invalid. (The standard routines take only one column name.)</td>
</tr>
<tr>
<td>missingfunction</td>
<td>User function that you specified does not exist.</td>
</tr>
<tr>
<td>missingconstraint</td>
<td>Constraint that you specified for a uniqueness conflict does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication group to which the specified table belongs is not quiesced.</td>
</tr>
<tr>
<td>duplicateresolution</td>
<td>Specified conflict resolution method is already registered.</td>
</tr>
<tr>
<td>duplicatesequence</td>
<td>The specified sequence number already exists for the specified object.</td>
</tr>
<tr>
<td>invalidprioritygroup</td>
<td>The specified priority group does not exist.</td>
</tr>
<tr>
<td>paramtype</td>
<td>Type is different from the type assigned to the priority group.</td>
</tr>
</tbody>
</table>
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 48-11

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 48–15 ALTER_CATCHUP_PARAMETERS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension_id</td>
<td>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.</td>
</tr>
<tr>
<td>percentage_for_catchup_mdef</td>
<td>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.</td>
</tr>
<tr>
<td>cycle_seconds_mdef</td>
<td>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.</td>
</tr>
<tr>
<td>percentage_for_catchup_new</td>
<td>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.</td>
</tr>
<tr>
<td>cycle_seconds_new</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–16 ALTER_CATCHUP_PARAMETERS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typefailure</td>
<td>The parameter value specified for one of the parameters is not appropriate.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group to which to alter the propagation mode.</td>
</tr>
<tr>
<td>master</td>
<td>Name of the master site at which to alter the propagation mode.</td>
</tr>
<tr>
<td>dblink_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>dblink_table</td>
<td>A PL/SQL index-by table, indexed from position 1, of database links for which to alter propagation.</td>
</tr>
<tr>
<td>propagation_mode</td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>This comment is added to the DBA_REPPROP view.</td>
</tr>
</tbody>
</table>
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Invocation site is not quiesced.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Propagation mode specified was not recognized.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>List of database links includes a site that is not a master site.</td>
</tr>
</tbody>
</table>

**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**

```sql
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 48–19 ALTER_MASTER_REPOBJECT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema containing the object that you want to alter.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you want to alter. The object cannot be a storage table for a nested table.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object that you are altering. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>FUNCTION SYNONYM</td>
</tr>
<tr>
<td></td>
<td>INDEX TABLE</td>
</tr>
<tr>
<td></td>
<td>INDEXTYPE TRIGGER</td>
</tr>
<tr>
<td></td>
<td>OPERATOR TYPE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE TYPE BODY</td>
</tr>
<tr>
<td></td>
<td>PACKAGE BODY VIEW</td>
</tr>
<tr>
<td></td>
<td>PROCEDURE</td>
</tr>
<tr>
<td>ddl_text</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>If not NULL, then this comment is added to the COMMENT field of the DBA_REPOBJECT view.</td>
</tr>
<tr>
<td>retry</td>
<td>If retry is true, then ALTER_MASTER_REPOBJECT alters the object only at masters whose object status is not VALID.</td>
</tr>
</tbody>
</table>
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.
ALTER_MVIEW_PROPAGATION Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Associated replication group has not been suspended.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Object identified by sname and oname does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
<tr>
<td>ddlfailure</td>
<td>DDL at the master definition site did not succeed.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.ALTER_MVIEW_PROPAGATION (
  gname IN VARCHAR2,
  propagation_mode IN VARCHAR2,
  comment IN VARCHAR2 := ''
  gowner IN VARCHAR2 := 'PUBLIC');
```
Parameters

Table 48–21  ALTER_MVIEW_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group for which to alter the propagation method.</td>
</tr>
<tr>
<td>propagation_mode</td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>This comment is added to the DBA_REPPROP view.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–22  ALTER_MVIEW_PROPAGATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingrepgroup</td>
<td>Specified replication group does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Propagation mode was specified incorrectly.</td>
</tr>
<tr>
<td>nonmview</td>
<td>Current site is not a materialized view site for the specified replication group.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Cannot contact master site or master materialized view site.</td>
</tr>
<tr>
<td>notcompat</td>
<td>Compatibility mode must be 7.3.0.0 or greater.</td>
</tr>
<tr>
<td>failaltermviewrop</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the priority group is associated.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group containing the priority that you want to alter.</td>
</tr>
<tr>
<td>old_priority</td>
<td>Current priority level of the priority group member.</td>
</tr>
<tr>
<td>new_priority</td>
<td>New priority level that you want assigned to the priority group member.</td>
</tr>
</tbody>
</table>

48-34  Supplied PL/SQL Packages and Types Reference
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>duplicatepriority</td>
<td>New priority level already exists in the priority group.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingvalue</td>
<td>Value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_datatype.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>Specified priority group does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced.</td>
</tr>
</tbody>
</table>

**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**

```sql
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 48–25  ALTER_PRIORITY_datatype Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the priority group is associated.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group containing the value that you want to alter.</td>
</tr>
<tr>
<td>old_value</td>
<td>Current value of the priority group member.</td>
</tr>
<tr>
<td>new_value</td>
<td>New value that you want assigned to the priority group member.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–26  ALTER_PRIORITY_datatype Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>duplicatevalue</td>
<td>New value already exists in the priority group.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>Specified priority group does not exist.</td>
</tr>
<tr>
<td>missingvalue</td>
<td>Old value does not exist.</td>
</tr>
<tr>
<td>paramtype</td>
<td>New value has the incorrect datatype for the priority group.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified value has the incorrect datatype for the priority group.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced.</td>
</tr>
</tbody>
</table>
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 48-27 ALTER_SITE_PRIORITY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the site priority group is associated.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group whose member you are altering.</td>
</tr>
<tr>
<td>old_priority</td>
<td>Current priority level of the site whose priority level you want to change.</td>
</tr>
<tr>
<td>new_priority</td>
<td>New priority level for the site. A higher number indicates a higher priority level.</td>
</tr>
</tbody>
</table>
ALTER_SITE_PRIORITY Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingpriority</td>
<td>Old priority level is not associated with any group members.</td>
</tr>
<tr>
<td>duplicatepriority</td>
<td>New priority level already exists for another site in the group.</td>
</tr>
<tr>
<td>missingvalue</td>
<td>Old value does not already exist.</td>
</tr>
<tr>
<td>paramtype</td>
<td>New value has the incorrect datatype for the priority group.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE (  
gname IN VARCHAR2,  
name IN VARCHAR2,  
old_site IN VARCHAR2,  
new_site IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the site priority group is associated.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group whose member you are altering.</td>
</tr>
<tr>
<td>old_site</td>
<td>Current global database name of the site to disassociate from the priority level.</td>
</tr>
<tr>
<td>new_site</td>
<td>New global database name that you want to associate with the current priority level.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingpriority</td>
<td>Specified site priority group does not exist.</td>
</tr>
<tr>
<td>missingvalue</td>
<td>Old site is not a group member.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced</td>
</tr>
</tbody>
</table>
CANCEL_STATISTICS Procedure

This procedure stops the collection of statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

Syntax

```sql
DBMS_REPCAT.CANCEL_STATISTICS (sname IN VARCHAR2,
oname IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table for which you do not want to gather conflict resolution statistics.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>statnotreg</td>
<td>Specified table is not currently registered to collect statistics.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP (
    sname IN VARCHAR2,
    oname IN VARCHAR2,
    column_group IN VARCHAR2,
    comment IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table with which the column group is associated.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group.</td>
</tr>
<tr>
<td>comment</td>
<td>Text of the updated comment that you want included in the GROUP_COMMENT field of the DBA_REPCOLUMN_GROUP view.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missinggroup</td>
<td>Specified column group does not exist.</td>
</tr>
<tr>
<td>missingobj</td>
<td>Object is missing.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.COMMENT_ON_MVIEW_REPSITES (  
gowner IN VARCHAR2,  
gname IN VARCHAR2,  
comment IN VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
<tr>
<td>gname</td>
<td>Name of the materialized view group.</td>
</tr>
<tr>
<td>comment</td>
<td>Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.</td>
</tr>
</tbody>
</table>

Table 48–36  COMMENT_ON_MVIEW_REPSITES Procedure Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingrepgroup</td>
<td>The materialized view group does not exist.</td>
</tr>
<tr>
<td>nonmview</td>
<td>The connected site is not a materialized view site.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group.</td>
</tr>
<tr>
<td>pgroup/name</td>
<td>Name of the priority or site priority group.</td>
</tr>
<tr>
<td>comment</td>
<td>Text of the updated comment that you want included in the</td>
</tr>
<tr>
<td></td>
<td>PRIORITY_COMMENT field of the DBA_REPPRIORITY_GROUP view.</td>
</tr>
</tbody>
</table>
COMMENT_ON_REPGROUP Procedure

Exceptions

<table>
<thead>
<tr>
<th>Table 48–38 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exception</strong></td>
</tr>
<tr>
<td>nonmasterdef</td>
</tr>
<tr>
<td>missingrepgroup</td>
</tr>
<tr>
<td>missingprioritygroup</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.COMMENT_ON_REPGROUP ( 
    gname IN VARCHAR2, 
    comment IN VARCHAR2) ;
```

Parameters

<table>
<thead>
<tr>
<th>Table 48–39 COMMENT_ON_REPGROUP Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>gname</td>
</tr>
<tr>
<td>comment</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Table 48–40 COMMENT_ON_REPGROUP Procedure Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exception</strong></td>
</tr>
<tr>
<td>nonmasterdef</td>
</tr>
<tr>
<td>commfailure</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.COMMENT_ON_REPOBJECT (sname IN VARCHAR2,
oname IN VARCHAR2,
type IN VARCHAR2,
comment IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you want to comment on. The object cannot be a storage table for a nested table.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object. The following types are supported: FUNCTION SYNONYM INDEX TABLE INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE</td>
</tr>
<tr>
<td>comment</td>
<td>Text of the updated comment that you want to include in the OBJECT_COMMENT field of the DBA_REPOBJECT view.</td>
</tr>
</tbody>
</table>
COMMENT_ON_REPSITES Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
</tbody>
</table>

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 48-48 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

```sql
DBMS_REPCAT.COMMENT_ON_REPSITES (gname IN VARCHAR2,
                                    [ master IN VARCHAR,]
                                    comment IN VARCHAR2);
```
Parameters

Table 48–43 COMMENT_ON_REPSITES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group. This avoids confusion if a database is a master site in more than one replication environment.</td>
</tr>
<tr>
<td>master</td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–44  COMMENT_ON_REPSITES Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>Invocation site is not a master site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group does not exist.</td>
</tr>
<tr>
<td>commfailure</td>
<td>One or more master sites are not accessible.</td>
</tr>
<tr>
<td>corrupt</td>
<td>There is an inconsistency in the replication catalog views.</td>
</tr>
</tbody>
</table>
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 48–45 COMMENT_ON_conflicttype_RESOLUTION Procedures

<table>
<thead>
<tr>
<th>Conflict Type</th>
<th>Procedure Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>COMMENT_ON_UPDATE_RESOLUTION</td>
</tr>
<tr>
<td>uniqueness</td>
<td>COMMENT_ON_UNIQUE_RESOLUTION</td>
</tr>
<tr>
<td>delete</td>
<td>COMMENT_ON_DELETE_RESOLUTION</td>
</tr>
</tbody>
</table>

The comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

Syntax

```sql
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 48–46  COMMENT_ON_conflicttype_RESOLUTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table with which the conflict resolution routine is associated.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group with which the update conflict resolution routine is associated.</td>
</tr>
<tr>
<td>constraint_name</td>
<td>Name of the unique constraint with which the uniqueness conflict resolution routine is associated.</td>
</tr>
<tr>
<td>sequence_no</td>
<td>Sequence number of the conflict resolution procedure.</td>
</tr>
<tr>
<td>comment</td>
<td>The text of the updated comment that you want included in the RESOLUTION_COMMENT field of the DBA_REPRESOLUTION view.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–47  COMMENT_ON_conflicttype_RESOLUTION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist.</td>
</tr>
<tr>
<td>missingresolution</td>
<td>Specified conflict resolution routine is not registered.</td>
</tr>
</tbody>
</table>
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.COMPAR_E_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

```sql
DBMS_REPCAT.COMPAR_E_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 48–48  COMPARE_OLD_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table. The table can be the storage table of a nested table.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated list of the columns in the table. There must be no spaces between entries.</td>
</tr>
<tr>
<td>column_table</td>
<td>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.</td>
</tr>
<tr>
<td>operation</td>
<td>Possible values are: update, delete, or the asterisk wildcard '*', which means update and delete.</td>
</tr>
<tr>
<td>compare</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
CREATE_MASTER_REPGROUP Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist as a table in the specified schema waiting for row-level replication information.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>At least one column is not in the table.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group has not been quiesced.</td>
</tr>
<tr>
<td>typefailure</td>
<td>An illegal operation is specified.</td>
</tr>
<tr>
<td>keysendcomp</td>
<td>A specified column is a key column in a table.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>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 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>

CREATE_MASTER_REPGROUP Procedure

This procedure creates a new, empty, quiesced master group.

Syntax

```sql
DBMS_REPCAT.CREATE_MASTER_REPGROUP (
    gname           IN  VARCHAR2,
    group_comment   IN  VARCHAR2  := '' ,
    master_comment  IN  VARCHAR2  := '' ,
    qualifier       IN  VARCHAR2  := ''
);
```
Parameters

**Table 48–50 CREATE_MASTER_REPGROUP Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group that you want to create.</td>
</tr>
<tr>
<td>group_comment</td>
<td>This comment is added to the DBA_REPGROUP view.</td>
</tr>
<tr>
<td>master_comment</td>
<td>This comment is added to the DBA_REPSITES view.</td>
</tr>
<tr>
<td>qualifier</td>
<td>Connection qualifier for master group. Be sure to use the @ sign.</td>
</tr>
<tr>
<td></td>
<td>See Oracle9i Replication and Oracle9i Database Administrator’s Guide for</td>
</tr>
<tr>
<td></td>
<td>more information about connection qualifiers.</td>
</tr>
</tbody>
</table>

Exceptions

**Table 48–51 CREATE_MASTER_REPGROUP Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicaterepgroup</td>
<td>Master group already exists.</td>
</tr>
<tr>
<td>norepopt</td>
<td>Advanced replication option is not installed.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Master group name was not specified.</td>
</tr>
<tr>
<td>qualifiertoolong</td>
<td>Connection qualifier is too long.</td>
</tr>
</tbody>
</table>

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.
CREATE_MASTER_REPOBJECT Procedure

Syntax

```sql
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>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the object that you want to replicate is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object you are replicating. If <code>ddl_text</code> is <code>NULL</code>, 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 long. The object cannot be a storage table for a nested table.</td>
</tr>
</tbody>
</table>
| type               | Type of the object that you are replicating. The following types are supported:  
                      | FUNCTION | SYNONYM  
                      | INDEX    | TABLE   
                      | INDEXTYPE | TRIGGER 
                      | OPERATOR  | TYPE    
                      | PACKAGE   | TYPE BODY |
                      | PACKAGE BODY | VIEW    |
                      | PROCEDURE  |         |
Table 48–52  CREATE_MASTER_REPOBJECT Procedure Parameters  (Page 2 of 2)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_existing_object</td>
<td>Indicate true if you want to reuse any objects of the same type and shape at the current master sites. See Table 48–54 for more information.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This parameter must be set to true for clustered tables.</td>
</tr>
<tr>
<td>ddl_text</td>
<td>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. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>comment</td>
<td>This comment is added to the OBJECT_COMMENT field of the DBA_REPOBJECT view.</td>
</tr>
<tr>
<td>retry</td>
<td>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.</td>
</tr>
<tr>
<td>copy_rows</td>
<td>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 48–54 for more information.</td>
</tr>
<tr>
<td>gname</td>
<td>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.</td>
</tr>
</tbody>
</table>
CREATE_MASTER_REPOBJECT Procedure

### Table 48–53 CREATE_MASTER_REPOBJECT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
<tr>
<td>duplicateobject</td>
<td>Specified object already exists in the master group and retry is false, or if a name conflict occurs.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Object identified by sname and oname does not exist and appropriate DDL has not been provided.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Objects of the specified type cannot be replicated.</td>
</tr>
<tr>
<td>ddlfailure</td>
<td>DDL at the master definition site did not succeed.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
<tr>
<td>notcompat</td>
<td>Not all remote masters in at least 7.3 compatibility mode.</td>
</tr>
</tbody>
</table>

### Object Creations

### Table 48–54 Object Creation at Master Sites

<table>
<thead>
<tr>
<th>Object</th>
<th>Already Exists?</th>
<th>COPY_ROWS</th>
<th>USE_EXISTING_OBJECTS</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>duplicatedobject message if objects do not match. For tables, use data from master definition site.</td>
</tr>
<tr>
<td>yes</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>duplicatedobject message if objects do not match. For tables, DBA must ensure contents are identical.</td>
</tr>
<tr>
<td>yes</td>
<td>true/false</td>
<td>false</td>
<td>true/false</td>
<td>duplicatedobject message.</td>
</tr>
<tr>
<td>no</td>
<td>true</td>
<td>true/false</td>
<td>true/false</td>
<td>Object is created. Tables populated using data from master definition site.</td>
</tr>
<tr>
<td>no</td>
<td>false</td>
<td>true/false</td>
<td>true/false</td>
<td>Object is created. DBA must populate tables and ensure consistency of tables at all sites.</td>
</tr>
</tbody>
</table>
CREATE_MVIEW_REPGROUP Procedure

This procedure creates a new, empty materialized view group in your local database. CREATE_MVIEW_REPGROUP automatically calls REGISTER_MVIEW_REPGROUP, but ignores any errors that may have happened during registration.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group. This group must exist at the specified master site or master materialized view site.</td>
</tr>
<tr>
<td>master</td>
<td>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.</td>
</tr>
<tr>
<td>comment</td>
<td>This comment is added to the DBA_REPGROUP view.</td>
</tr>
<tr>
<td>propagation_mode</td>
<td>Method of propagation for all updatable materialized views in the replication group. Acceptable values are synchronous and asynchronous.</td>
</tr>
<tr>
<td>fname</td>
<td>This parameter is for internal use only.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

Note: Do not set this parameter unless directed to do so by Oracle Support Services.
CREATE_MVIEW_REPOBJECT Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicaterepgroup</td>
<td>Replication group already exists at the invocation site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>Specified database is not a master site or master materialized view site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Specified database is not accessible.</td>
</tr>
<tr>
<td>norepopt</td>
<td>Advanced replication option is not installed.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Propagation mode was specified incorrectly.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group does not exist at master site.</td>
</tr>
<tr>
<td>invalidqualifier</td>
<td>Connection qualifier specified for the master site or master materialized view site is not valid for the replication group.</td>
</tr>
<tr>
<td>alreadymastered</td>
<td>At the local site, there is another materialized view group with the same group name, but different master site or master materialized view site.</td>
</tr>
</tbody>
</table>

CREATE_MVIEW_REPOBJECT Procedure

This procedure adds a replicated object to a materialized view group.

Syntax

```sql
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 48–57  CREATE_MVIEW_REPOBJECT Procedure Parameters  (Page 1 of 2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>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.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you want to add to the replicated materialized view group.</td>
</tr>
<tr>
<td>type</td>
<td>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</td>
</tr>
<tr>
<td>ddl_text</td>
<td>For objects of type SNAPSHOT, the DDL needed to create the object. For other types, use the default: ‘’ (an empty string)</td>
</tr>
</tbody>
</table>

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.
This comment is added to the OBJECTCOMMENT field of the DBA_REPOBJECT view.

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.

Name of the user you want to assign as owner of the transaction.

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.

Set to true if the materialized view’s master site is running a version of Oracle server prior to Oracle8i release 8.1.5. Set to false if the materialized view’s master site or master materialized view site is running Oracle8i release 8.1.5 or greater.

Owner of the materialized view group.
Exceptions

Table 48–58 CREATE_MVIEW_REPOBJECT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>Master is no longer a master site or master materialized view site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist in the master’s replication group.</td>
</tr>
<tr>
<td>duplicateobject</td>
<td>Specified object already exists with a different shape.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Type is not an allowable type.</td>
</tr>
<tr>
<td>ddlfailure</td>
<td>DDL did not succeed.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Master site or master materialized view site is not accessible.</td>
</tr>
<tr>
<td>missingschema</td>
<td>Schema does not exist as a database schema.</td>
</tr>
<tr>
<td>badmviewddl</td>
<td>DDL was executed but materialized view does not exist.</td>
</tr>
<tr>
<td>onlyonemview</td>
<td>Only one materialized view for master table or master materialized view can be created.</td>
</tr>
<tr>
<td>badmviewname</td>
<td>Materialized view base table differs from master table or master materialized view.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group at the master does not exist.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_REPCAT.DEFINE_COLUMN_GROUP (sname IN VARCHAR2,
oname IN VARCHAR2,
column_group IN VARCHAR2,
comment IN VARCHAR2 := NULL);
```
DEFINE_PRIORITY_GROUP Procedure

Parameters

Table 48–59  DEFINE_COLUMN_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table for which you are creating a column group.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group that you want to create.</td>
</tr>
<tr>
<td>comment</td>
<td>This user text is displayed in the DBA_REPCOLUMN_GROUP view.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–60  DEFINE_COLUMN_GROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>duplicategroup</td>
<td>Specified column group already exists for the table.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication group to which the specified table belongs is not quiesced.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.DEFINE_PRIORITY_GROUP (  
  gname  IN VARCHAR2,  
  pgroup  IN VARCHAR2,  
  datatype  IN VARCHAR2,  
  fixed_length  IN INTEGER := NULL,  
  comment  IN VARCHAR2 := NULL);
```

48-62  Supplied PL/SQL Packages and Types Reference
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group for which you are creating a priority group.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group that you are creating.</td>
</tr>
<tr>
<td>datatype</td>
<td>Datatype of the priority group members. The datatypes supported are: CHAR, VARCHAR2, NUMBER, DATE, RAW, NCHAR, and NVARCHAR2.</td>
</tr>
<tr>
<td>fixed_length</td>
<td>You must provide a column length for the CHAR datatype. All other types can use the default, NULL.</td>
</tr>
<tr>
<td>comment</td>
<td>This user comment is added to the DBA_REPPRIORITY view.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>duplicatepriority</td>
<td>Specified priority group already exists in the master group.</td>
</tr>
<tr>
<td>group</td>
<td>Specified datatype is not supported.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_REPCAT.DEFINE_SITE_PRIORITY (gname IN VARCHAR2,
                        name  IN VARCHAR2,
                        comment IN VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>The master group for which you are creating a site priority group.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group that you are creating.</td>
</tr>
<tr>
<td>comment</td>
<td>This user comment is added to the DBA_REPPRIORITY view.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>duplicate</td>
<td>Specified site priority group already exists in the master group.</td>
</tr>
<tr>
<td>prioritygroup</td>
<td></td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN ( 
    gname IN VARCHAR2,
    all_sites IN BOOLEAN := false);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group.</td>
</tr>
<tr>
<td>all_sites</td>
<td>If this is true, then use a job to execute the local administrative procedures at each master site.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>Invocation site is not a master site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible and all_sites is true.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table whose column group you are dropping.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group that you want to drop.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>referenced</td>
<td>Specified column group is being used in conflict detection and resolution.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>missinggroup</td>
<td>Specified column group does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group to which the table belongs is not quiesced.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table in which the column group is located. The table can be the storage table of a nested table.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group from which you are removing members.</td>
</tr>
<tr>
<td>list_of_column_names</td>
<td>Names of the columns that you are removing from the designated column group. This can either be a comma-separated 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.</td>
</tr>
</tbody>
</table>
DROP_MASTER_REPGROUP Procedure

Exceptions

Table 48–70 DROP_GROUPED_COLUMN Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group that the table belongs to is not quiesced.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.DROP_MASTER_REPGROUP (
  gname IN VARCHAR2,
  drop_contents IN BOOLEAN := false,
  all_sites IN BOOLEAN := false);
```

Parameters

Table 48–71 DROP_MASTER_REPGROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group that you want to drop from the current master site.</td>
</tr>
<tr>
<td>drop_contents</td>
<td>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.</td>
</tr>
<tr>
<td>all_sites</td>
<td>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.</td>
</tr>
</tbody>
</table>
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>Invocation site is not a master site.</td>
</tr>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site and all_sites is true.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible and all_sites is true.</td>
</tr>
<tr>
<td>fullqueue</td>
<td>Deferred remote procedure call (RPC) queue has entries for the master group.</td>
</tr>
<tr>
<td>masternotremoved</td>
<td>Master does not recognize the master definition site and all_sites is true.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_REPCAT.DROP_MASTER_REPOBJECT (sname IN VARCHAR2, oname IN VARCHAR2, type IN VARCHAR2, drop_objects IN BOOLEAN := false);
```
Parameters

Table 48–73  DROP_MASTER_REPOBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you want to remove from the master group. The object cannot be a storage table for a nested table.</td>
</tr>
<tr>
<td>type</td>
<td>Type of object that you want to drop. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>FUNCTION SYNONYM</td>
</tr>
<tr>
<td></td>
<td>INDEX TABLE</td>
</tr>
<tr>
<td></td>
<td>INDEXTYPE TRIGGER</td>
</tr>
<tr>
<td></td>
<td>OPERATOR TYPE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE TYPE BODY</td>
</tr>
<tr>
<td></td>
<td>PACKAGE BODY VIEW</td>
</tr>
<tr>
<td>drop_objects</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–74  DROP_MASTER_REPOBJECT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.DROP_MVIEW_REPGROUP (  
gname IN VARCHAR2,  
drop_contents IN BOOLEAN := false  
gowner IN VARCHAR2 := 'PUBLIC');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>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.</td>
</tr>
<tr>
<td>drop_contents</td>
<td>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.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified replication group does not exist.</td>
</tr>
</tbody>
</table>
DROP_MVIEW_REPOBJECT Procedure

This procedure drops a replicated object from a materialized view site.

Syntax

```sql
DBMS_REPCAT.DROP_MVIEW_REPOBJECT (  
sname IN VARCHAR2,  
oname IN VARCHAR2,  
type IN VARCHAR2,  
drop_objects IN BOOLEAN := false);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you want to drop from the replication group.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object that you want to drop. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>FUNCTION SYNCHRONOUS</td>
</tr>
<tr>
<td></td>
<td>INDEX SYNONYM</td>
</tr>
<tr>
<td></td>
<td>INDEXTYPE TRIGGER</td>
</tr>
<tr>
<td></td>
<td>OPERATOR TYPE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE TYPE BODY</td>
</tr>
<tr>
<td></td>
<td>PACKAGE BODY VIEW</td>
</tr>
<tr>
<td></td>
<td>PROCEDURE</td>
</tr>
<tr>
<td>drop_objects</td>
<td>By default, the object remains in its associated schema, but is dropped</td>
</tr>
<tr>
<td></td>
<td>from its associated replication group. To completely remove the object from</td>
</tr>
<tr>
<td></td>
<td>its schema at the current materialized view site, set this parameter to</td>
</tr>
<tr>
<td></td>
<td>true. If the parameter is set to true, the object is dropped from the</td>
</tr>
<tr>
<td></td>
<td>database at the materialized view site.</td>
</tr>
</tbody>
</table>

48-72  Supplied PL/SQL Packages and Types Reference
Exceptions

Table 48–78 DROP_MVIEW_REPOBJECT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.DROP_PRIORITY(
    gname IN VARCHAR2,
    pgroup IN VARCHAR2,
    priority_num IN NUMBER);
```

Parameters

Table 48–79 DROP_PRIORITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the priority group is associated.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group containing the member that you want to drop.</td>
</tr>
<tr>
<td>priority_num</td>
<td>Priority level of the priority group member that you want to remove from the group.</td>
</tr>
</tbody>
</table>
DROP_PRIORITY_GROUP Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>Specified priority group does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the priority group is associated.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group that you want to drop.</td>
</tr>
</tbody>
</table>
Exceptions

Table 48–82  DROP_PRIORITY_GROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>referenced</td>
<td>Specified priority group is being used in conflict resolution.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_REPCAT.DROP_PRIORITY_\_datatype (  
gname IN VARCHAR2,  
pgroup IN VARCHAR2,  
value IN \_datatype);  
```

where _datatype:_

```sql
(  NUMBER  
  | VARCHAR2  
  | CHAR  
  | DATE  
  | RAW  
  | NCHAR  
  | NVARCHAR2 )
```
DROP_SITE_PRIORITY Procedure

Parameters

Table 48–83  DROP_PRIORITY_datatype Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the priority group is associated.</td>
</tr>
<tr>
<td>pgroup</td>
<td>Name of the priority group containing the member that you want to drop.</td>
</tr>
<tr>
<td>value</td>
<td>Value of the priority group member that you want to remove from the group.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–84  DROP_PRIORITY_datatype Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingprioritygroup</td>
<td>Specified priority group does not exist.</td>
</tr>
<tr>
<td>paramtype,</td>
<td>Value has the incorrect datatype for the priority group.</td>
</tr>
<tr>
<td>typefailure</td>
<td></td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced</td>
</tr>
</tbody>
</table>

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 48–85  DROP_SITE_PRIORITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the site priority group is associated.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group that you want to drop.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–86  DROP_SITE_PRIORITY Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>referenced</td>
<td>Specified site priority group is being used in conflict resolution.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced</td>
</tr>
</tbody>
</table>

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);  
```
DROP_conflicttype_RESOLUTION Procedure

Parameters

Table 48–87  DROP_SITE_PRIORITY_SITE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group with which the site priority group is associated.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the site priority group whose member you are dropping.</td>
</tr>
<tr>
<td>site</td>
<td>Global database name of the site you are removing from the group.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–88  DROP_SITE_PRIORITY_SITE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Specified master group does not exist.</td>
</tr>
<tr>
<td>missingpriority</td>
<td>Specified site priority group does not exist.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Specified master group is not quiesced.</td>
</tr>
</tbody>
</table>

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 48–89  Conflict Resolution Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Procedure Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>DROP_UPDATE_RESOLUTION</td>
</tr>
<tr>
<td>uniqueness</td>
<td>DROP_UNIQUE_RESOLUTION</td>
</tr>
<tr>
<td>delete</td>
<td>DROP_DELETE_RESOLUTION</td>
</tr>
</tbody>
</table>
Summary of DBMS_REPCAT Subprograms

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table for which you want to drop a conflict resolution routine.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name of the column group for which you want to drop an update conflict resolution routine.</td>
</tr>
<tr>
<td>constraint_name</td>
<td>Name of the unique constraint for which you want to drop a unique conflict resolution routine.</td>
</tr>
<tr>
<td>sequence_no</td>
<td>Sequence number assigned to the conflict resolution method that you want to drop. This number uniquely identifies the routine.</td>
</tr>
</tbody>
</table>
EXECUTE_DDL Procedure

Exceptions

Table 48–91 DROP_conflicttype_RESOLUTION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist as a table in the specified schema, or a</td>
</tr>
<tr>
<td></td>
<td>conflict resolution routine with the specified sequence number is not</td>
</tr>
<tr>
<td></td>
<td>registered.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>

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

```sql
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 48–92  EXECUTE_DDL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group.</td>
</tr>
<tr>
<td>master_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>master_table</td>
<td>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.</td>
</tr>
<tr>
<td>ddl_text</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–93  EXECUTE_DDL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>At least one site is not a master site.</td>
</tr>
<tr>
<td>ddlfailure</td>
<td>DDL at the master definition site did not succeed.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
</tbody>
</table>
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>The name of the object for which you are generating support.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object. The types supported are SNAPSHOT, PACKAGE, and PACKAGE BODY.</td>
</tr>
<tr>
<td>gen_objs_owner</td>
<td>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.</td>
</tr>
<tr>
<td>min_communication</td>
<td>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.</td>
</tr>
<tr>
<td>generate_80_compatible</td>
<td>Set to true if the materialized view’s master site is running a version of Oracle server prior to Oracle8i release 8.1.5. Set to false if the materialized view’s master site or master materialized view site is running Oracle8i release 8.1.5 or higher.</td>
</tr>
</tbody>
</table>
Exceptions

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>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.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
<tr>
<td>missingschema</td>
<td>Specified owner of generated objects does not exist.</td>
</tr>
<tr>
<td>missingremoteobject</td>
<td>Object at master site or master materialized view site has not yet generated replication support.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Master site or master materialized view site is not accessible.</td>
</tr>
</tbody>
</table>

**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**

```sql
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,  
  genobjs_owner IN VARCHAR2 := NULL,  
  min_communication IN BOOLEAN := true,  
  generate_80_compatible IN BOOLEAN := true);  
```
### Parameters

**Table 48-96  GENERATE_REPLICATION_SUPPORT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the object is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object for which you are generating replication support.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object. The types supported are: TABLE, PACKAGE, and PACKAGE BODY.</td>
</tr>
<tr>
<td>package_prefix</td>
<td>For objects of type PACKAGE or PACKAGE BODY this value is prepended to the generated wrapper package name. The default is DEFER_.</td>
</tr>
<tr>
<td>procedure_prefix</td>
<td>For objects of type PACKAGE or PACKAGE BODY, this value is prepended to the generated wrapper procedure names. By default, no prefix is assigned.</td>
</tr>
<tr>
<td>distributed</td>
<td>This must be set to true.</td>
</tr>
<tr>
<td>gen_objs_owner</td>
<td>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.</td>
</tr>
<tr>
<td>min_communication</td>
<td>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.</td>
</tr>
<tr>
<td>generate_80_compatible</td>
<td>Set to true if any master site is running a version of Oracle server prior to Oracle8i release 8.1.5. Set to false if all master sites are running Oracle8i release 8.1.5 or higher.</td>
</tr>
</tbody>
</table>
Exceptions

Table 48–97  GENERATE_REPLICATION_SUPPORT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>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.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Specified type parameter is not supported.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication group has not been quiesced.</td>
</tr>
<tr>
<td>connfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
<tr>
<td>missingschema</td>
<td>Schema does not exist.</td>
</tr>
<tr>
<td>dnotcompatible</td>
<td>One of the master sites is not 7.3.0.0 compatible.</td>
</tr>
<tr>
<td>notcompat</td>
<td>One of the master sites is not 7.3.0.0 compatible. (Equivalent to dnotcompatible.)</td>
</tr>
<tr>
<td>duplicateobject</td>
<td>Object already exists.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.MAKE_COLUMN_GROUP ( 
    sname IN VARCHAR2, 
    oname IN VARCHAR2, 
    column_group IN VARCHAR2, 
    list_of_column_names IN VARCHAR2 | DBMS_REPCAT.VARCHAR2s);
```
MAKE_COLUMN_GROUP Procedure

Parameters

Table 48–98 MAKE_COLUMN_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>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.</td>
</tr>
<tr>
<td>column_group</td>
<td>Name that you want assigned to the column group that you are creating.</td>
</tr>
<tr>
<td>list_of_column_names</td>
<td>Names of the columns that you are grouping. This can either be a comma-separated 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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–99 MAKE_COLUMN_GROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the masterdef site.</td>
</tr>
<tr>
<td>duplicategroup</td>
<td>Specified column group already exists for the table.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>Specified column does not exist in the designated table.</td>
</tr>
<tr>
<td>duplicatecolumn</td>
<td>Specified column is already a member of another column group.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiesced.</td>
</tr>
</tbody>
</table>
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.
PURGE_MASTER_LOG Procedure

Syntax

```sql
DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (  
    extension_id  IN  RAW);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension_id</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typefailure</td>
<td>The parameter value specified for one of the parameters is not appropriate.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.PURGE_MASTER_LOG (  
    id IN  BINARY_INTEGER,  
    source IN  VARCHAR2,  
    gname IN  VARCHAR2);
```
**Parameters**

<table>
<thead>
<tr>
<th>Table 48–102</th>
<th>PURGE_MASTER_LOG Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>id</td>
<td>Identification number of the request, as it appears in the DBA_REPCATLOG view.</td>
</tr>
<tr>
<td>source</td>
<td>Master site from which the request originated.</td>
</tr>
<tr>
<td>gname</td>
<td>Name of the master group for which the request was made.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Table 48–103</th>
<th>PURGE_MASTER_LOG Procedure Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>nonmaster</td>
<td>gname is not NULL, and the invocation site is not a master site.</td>
</tr>
</tbody>
</table>

**PURGE_STATISTICS Procedure**

This procedure removes information from the DBA_REPRESOLUTION_STATISTICS view.

**Syntax**

```sql
DBMS_REPCAT.PURGE_STATISTICS (
    sname IN VARCHAR2,
    oname IN VARCHAR2,
    start_date IN DATE,
    end_date IN DATE);
```
REFRESH_MVIEW_REPGROUP Procedure

Parameters

Table 48–104  PURGE_STATISTICS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table whose conflict resolution statistics you want to purge.</td>
</tr>
<tr>
<td>start_date/end_date</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–105  PURGE_STATISTICS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
<tr>
<td>statnotreg</td>
<td>Table not registered to collect statistics.</td>
</tr>
</tbody>
</table>

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

```sql
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’);
```
Summary of DBMS_REPCAT Subprograms

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>gname</strong></td>
<td>Name of the replication group.</td>
</tr>
<tr>
<td><strong>drop_missing_contents</strong></td>
<td>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 <code>true</code>.</td>
</tr>
<tr>
<td><strong>refresh_mviews</strong></td>
<td>Set to <code>true</code> to refresh the contents of the materialized views in the replication group.</td>
</tr>
<tr>
<td><strong>refresh_other_objects</strong></td>
<td>Set this to <code>true</code> to refresh the contents of the nonmaterialized view objects in the replication group. Nonmaterialized view objects may include the following:</td>
</tr>
<tr>
<td></td>
<td>• Tables</td>
</tr>
<tr>
<td></td>
<td>• Views</td>
</tr>
<tr>
<td></td>
<td>• Indexes</td>
</tr>
<tr>
<td></td>
<td>• PL/SQL packages and package bodies</td>
</tr>
<tr>
<td></td>
<td>• PL/SQL procedures and functions</td>
</tr>
<tr>
<td></td>
<td>• Triggers</td>
</tr>
<tr>
<td></td>
<td>• Synonyms</td>
</tr>
<tr>
<td><strong>gowner</strong></td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>
REGISTER_MVIEW_REPGROUP Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>Master is no longer a master site or master</td>
</tr>
<tr>
<td></td>
<td>materialized view site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Master site or master materialized view site is not</td>
</tr>
<tr>
<td></td>
<td>accessible.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group name not specified.</td>
</tr>
</tbody>
</table>

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’);  
```
Summary of DBMS_REPCAT Subprograms

Parameters

Table 48–108  REGISTER_MVIEW_REPGROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the materialized view group to be registered.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>Global name of the materialized view site.</td>
</tr>
<tr>
<td>comment</td>
<td>Comment for the materialized view site or update for an existing comment.</td>
</tr>
<tr>
<td>rep_type</td>
<td>Version of the materialized view group. Valid constants that can be assigned include the following:</td>
</tr>
<tr>
<td></td>
<td>• dbms_repcat.reg_unknown (the default)</td>
</tr>
<tr>
<td></td>
<td>• dbms_repcat.reg_v7_group</td>
</tr>
<tr>
<td></td>
<td>• dbms_repcat.reg_v8_group</td>
</tr>
<tr>
<td>fname</td>
<td>This parameter is for internal use only.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

Note: Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 48–109  REGISTER_MVIEW_REPGROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failregmviewrepgroup</td>
<td>Registration of materialized view group failed.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Replication group name not specified.</td>
</tr>
<tr>
<td>nullsitename</td>
<td>A materialized view site was not specified.</td>
</tr>
<tr>
<td>normaster</td>
<td>Procedure must be executed at the materialized view’s master site or master materialized view site.</td>
</tr>
<tr>
<td>duplicaterepgroup</td>
<td>Replication group already exists.</td>
</tr>
</tbody>
</table>
REGISTER_STATISTICS Procedure

This procedure collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.

Syntax

```sql
DBMS_REPCAT.REGISTER_STATISTICS (
    sname IN VARCHAR2,
    oname IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Name of the schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table for which you want to gather conflict resolution statistics.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingschema</td>
<td>Specified schema does not exist.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified table does not exist.</td>
</tr>
</tbody>
</table>

48-94  Supplied PL/SQL Packages and Types Reference
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

```sql
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);
```
RELOCATE_MASTERDEF Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the replication group whose master definition you want to relocate.</td>
</tr>
<tr>
<td>old_masterdef</td>
<td>Fully qualified database name of the current master definition site.</td>
</tr>
<tr>
<td>new_masterdef</td>
<td>Fully qualified database name of the existing master site that you want to make the new master definition site.</td>
</tr>
<tr>
<td>notify_masters</td>
<td>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.</td>
</tr>
<tr>
<td>include_old_masterdef</td>
<td>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.</td>
</tr>
<tr>
<td>require_flavor_change</td>
<td>This parameter is for internal use only.</td>
</tr>
</tbody>
</table>

Note: Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>new_masterdef is not a master site or the invocation site is not a master site.</td>
</tr>
<tr>
<td>nonmasterdef</td>
<td>old_masterdef is not the master definition site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible and notify_masters is true.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>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.</td>
</tr>
<tr>
<td>master_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>master_table</td>
<td>In place of a list, you can specify the database names in a PL/SQL index-by table of type DBMS_UTILITY.DBLINK_ARRAY.</td>
</tr>
</tbody>
</table>
RENAME_SHADOW_COLUMN_GROUP Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>At least one of the specified databases is not a master site.</td>
</tr>
<tr>
<td>reconfigerror</td>
<td>One of the specified databases is the master definition site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one remaining master site is not accessible.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT.RENAME_SHADOW_COLUMN_GROUP (  
  sname IN VARCHAR2,  
  oname IN VARCHAR2,  
  new_col_group_name IN VARCHAR2)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the replicated table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table.</td>
</tr>
<tr>
<td>new_col_group_name</td>
<td>Name of the new column group. The columns currently in the shadow group are placed in a column group with the name you specify.</td>
</tr>
</tbody>
</table>
Exclusions

**Table 48–117 RENAME_SHADOW_COLUMN_GROUP Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missmview</td>
<td>The specified schema does not exist.</td>
</tr>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>The specified object does not exist.</td>
</tr>
<tr>
<td>duplicategroup</td>
<td>The column group that was specified for creation already exists.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_REPCAT.REPCAT_IMPORT_CHECK (  
  gname IN VARCHAR2,     
  master IN BOOLEAN,     
  gowner IN VARCHAR2 := 'PUBLIC');
```

**Parameters**

**Table 48–118 REPCAT_IMPORT_CHECK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group. If you omit both parameters, then the procedure checks all master groups at your current site.</td>
</tr>
<tr>
<td>master</td>
<td>Set this to true if you are checking a master site and false if you are checking a materialized view site.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the master group.</td>
</tr>
</tbody>
</table>
RESUME_MASTER_ACTIVITY Procedure

Exceptions

Table 48–119 REPCAT_IMPORT_CHECK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>master is true and either the database is not a master site for the replication group or the database is not the expected database.</td>
</tr>
<tr>
<td>nonmview</td>
<td>master is false and the database is not a materialized view site for the replication group.</td>
</tr>
<tr>
<td>missingobject</td>
<td>A valid replicated object in the replication group does not exist.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>The specified replicated replication group does not exist.</td>
</tr>
<tr>
<td>missingschema</td>
<td>The specified replicated replication group does not exist.</td>
</tr>
</tbody>
</table>

RESUME_MASTER_ACTIVITY Procedure

This procedure resumes normal replication activity after quiescing a replication environment.

Syntax

```sql
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (  
    gname IN VARCHAR2,  
    override IN BOOLEAN := false);
```

Parameters

Table 48–120 RESUME_MASTER_ACTIVITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group.</td>
</tr>
</tbody>
</table>
| 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>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group is not quiescing or quiesced.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
<tr>
<td>notallgenerated</td>
<td>Generate replication support before resuming replication activity.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
    extension_id IN RAW);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension_id</td>
<td>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.</td>
</tr>
</tbody>
</table>
SEND_OLD_VALUES Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>extstinapp</td>
<td>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.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```

48-102  Supplied PL/SQL Packages and Types Reference
Summary of DBMS_REPCAT Subprograms

---

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

---

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the replicated table. The table can be the storage table of a nested table.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated list of the columns in the table. There must be no spaces between entries.</td>
</tr>
<tr>
<td>column_table</td>
<td>Instead of a list, you can use a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMSUTILITY.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 DBMSUTILITY.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.</td>
</tr>
<tr>
<td>operation</td>
<td>Possible values are: update, delete, or the asterisk wildcard ‘*’, which means update and delete.</td>
</tr>
<tr>
<td>send</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

DBMS_REPCAT 48-103
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>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist as a table in the specified schema waiting for row-level replication information.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>At least one column is not in the table.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Master group has not been quiesced.</td>
</tr>
<tr>
<td>typefailure</td>
<td>An illegal operation is specified.</td>
</tr>
<tr>
<td>keysendcomp</td>
<td>A specified column is a key column in a table.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>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 9.0.0 or higher compatibility level.</td>
</tr>
</tbody>
</table>
**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**

```sql
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 } );
```
SET_COLUMNS Procedure

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

### Parameters

**Table 48–126 SET_COLUMNS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td>Schema in which the table is located.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-separated list of the columns in the table that you want to use as a primary key. There must be no spaces between entries.</td>
</tr>
<tr>
<td>column_table</td>
<td>Instead of a list, you can use a PL/SQL index-by table of type DBMS_UTILITY.LNAME_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.</td>
</tr>
</tbody>
</table>

### Exceptions

**Table 48–127 SET_COLUMNS Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>missingobject</td>
<td>Specified object does not exist as a table in the specified schema waiting for row-level replication information.</td>
</tr>
<tr>
<td>missingcolumn</td>
<td>At least one column is not in the table.</td>
</tr>
<tr>
<td>notquiesced</td>
<td>Replication group is not quiescing or quiesced.</td>
</tr>
</tbody>
</table>
**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 48-11

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Master group to which you are adding new master sites.</td>
</tr>
<tr>
<td>master_list</td>
<td>A comma-separated 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.</td>
</tr>
<tr>
<td>master_table</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicaterespgroup</td>
<td>A master site that you are attempting to add is already part of the master group.</td>
</tr>
<tr>
<td>normasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>propmodenotallowed</td>
<td>Synchronous propagation mode not allowed for this operation. Only asynchronous propagation mode is allowed.</td>
</tr>
<tr>
<td>extstininapp</td>
<td>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.</td>
</tr>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
<tr>
<td>notsamecq</td>
<td>Master groups do not have the same connection qualifier.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group for which you want to suspend activity.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmasterdef</td>
<td>Invocation site is not the master definition site.</td>
</tr>
<tr>
<td>notnormal</td>
<td>Master group is not in normal operation.</td>
</tr>
<tr>
<td>commfailure</td>
<td>At least one master site is not accessible.</td>
</tr>
</tbody>
</table>
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 Oracle8i (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 recreated if you want to base it on a different master.

**See Also:** "GENERATE_MVIEW_SUPPORT Procedure" on page 48-82

### Syntax

```sql
DBMS_REPCAT.SWITCH_MVIEW_MASTER (
  gname IN VARCHAR2,
  master IN VARCHAR2,
  gowner IN VARCHAR2 := 'PUBLIC');
```
Parameter

Table 48–132 SWITCH_MVIEW_MASTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the materialized view group for which you want to change the master site.</td>
</tr>
<tr>
<td>master</td>
<td>Fully qualified database name of the new master site to use for the materialized view group.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

Exceptions

Table 48–133 SWITCH_MVIEW_MASTER Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmview</td>
<td>Invocation site is not a materialized view site.</td>
</tr>
<tr>
<td>nonmaster</td>
<td>Specified database is not a master site.</td>
</tr>
<tr>
<td>commfailure</td>
<td>Specified database is not accessible.</td>
</tr>
<tr>
<td>missingrepgroup</td>
<td>Materialized view group does not exist.</td>
</tr>
<tr>
<td>qrytooolong</td>
<td>Materialized view definition query is greater 32 KB.</td>
</tr>
<tr>
<td>alreadymastered</td>
<td>At the local site, there is another materialized view group with the same group name mastered at the old master site.</td>
</tr>
</tbody>
</table>

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.
UNDO_ADD_NEW_MASTERS_REQUEST Procedure

**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 48-107
- "ADD_NEW_MASTERS Procedure" on page 48-11
- "RESUME_PROPAGATION_TO_MDEF Procedure" on page 48-101
- "PREPARE_INSTANTIATED_MASTER Procedure" on page 48-87

Syntax

```
DBMS_REPCAT.UNDO_ADD_NEW_MASTERS_REQUEST (  
    extension_id IN RAW,  
    drop_contents IN BOOLEAN := TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension_id</td>
<td>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.</td>
</tr>
<tr>
<td>drop_contents</td>
<td>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.</td>
</tr>
</tbody>
</table>
Exceptions

Table 48-135  UNDO_ADD_NEW_MASTERS_REQUEST Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbnotcompatible</td>
<td>Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.</td>
</tr>
<tr>
<td>typefail</td>
<td>A parameter value that you specified is not appropriate.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.UNREGISTER_MVIEW_REPGROUP (
    gname IN VARCHAR2,
    mviewsite IN VARCHAR2,
    gowner IN VARCHAR2 := 'PUBLIC');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the materialized view group to be unregistered.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>Global name of the materialized view site.</td>
</tr>
<tr>
<td>gowner</td>
<td>Owner of the materialized view group.</td>
</tr>
</tbody>
</table>

VALIDATE Function

This function validates the correctness of key conditions of a multimaster replication environment.

Syntax

```sql
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 48-114, 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 48-114 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 48–137  VALIDATE Function Parameters  (Page 1 of 2)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group to validate.</td>
</tr>
<tr>
<td>check_genflags</td>
<td>Check whether all the objects in the group are generated. This must be done at the master definition site only.</td>
</tr>
<tr>
<td>check_valid_objs</td>
<td>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.</td>
</tr>
<tr>
<td>check_links_sched</td>
<td>Check whether the links are scheduled for execution. This should be invoked at each master site.</td>
</tr>
</tbody>
</table>
Exceptions

Table 48–138 VALIDATE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missingdblink</td>
<td>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.</td>
</tr>
<tr>
<td>dblinkmismatch</td>
<td>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.</td>
</tr>
<tr>
<td>dblinkuidmismatch</td>
<td>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.</td>
</tr>
<tr>
<td>objectnotgenerated</td>
<td>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.</td>
</tr>
<tr>
<td>opnotsupported</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT.WAIT_MASTER_LOG (  
gname IN VARCHAR2,  
record_count IN NATURAL,  
timeout IN NATURAL,  
true_count OUT NATURAL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>Name of the master group.</td>
</tr>
<tr>
<td>record_count</td>
<td>Procedure returns whenever the number of incomplete activities is at or below this threshold.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum number of seconds to wait before the procedure returns.</td>
</tr>
<tr>
<td>true_count</td>
<td>Returns the number of incomplete activities.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>Invocation site is not a master site.</td>
</tr>
</tbody>
</table>
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
## Summary of DBMS_REPCAT_ADMIN Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GRANT_ADMIN_ANY_SCHEMA Procedure&quot; on page 49-3</td>
<td>Grants the necessary privileges to the replication administrator to administer any replication group at the current site.</td>
</tr>
<tr>
<td>&quot;GRANT_ADMIN_SCHEMA Procedure&quot; on page 49-4</td>
<td>Grants the necessary privileges to the replication administrator to administer a schema at the current site.</td>
</tr>
<tr>
<td>&quot;REGISTER_USER_REPGROUP Procedure&quot; on page 5</td>
<td>Assigns proxy materialized view administrator or receiver privileges at the master site or master materialized view site for use with remote sites.</td>
</tr>
<tr>
<td>&quot;REVOKE_ADMIN_ANY_SCHEMA Procedure&quot; on page 49-7</td>
<td>Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_ANY_SCHEMA.</td>
</tr>
<tr>
<td>&quot;REVOKE_ADMIN_SCHEMA Procedure&quot; on page 49-8</td>
<td>Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_SCHEMA.</td>
</tr>
<tr>
<td>&quot;UNREGISTER_USER_REPGROUP Procedure&quot; on page 49-9</td>
<td>Revokes the privileges and roles from the proxy materialized view administrator or receiver that were granted by the REGISTER_USER_REPGROUP procedure.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (username IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
</tbody>
</table>
**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**

```sql
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.
REGISTER_USER_REPGROUP Procedure

Parameters

Table 49–6  REGISTER_USER_REPGROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user to whom you are giving either proxy materialized view administrator or receiver privileges.</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specifies the privilege type you are assigning. Use the following values for to define your privilege_type:</td>
</tr>
<tr>
<td></td>
<td>- receiver for receiver privileges</td>
</tr>
<tr>
<td></td>
<td>- proxy_snapadmin for proxy materialized view administration privileges</td>
</tr>
<tr>
<td>list_of_gnames</td>
<td>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.</td>
</tr>
<tr>
<td>table_of_gnames</td>
<td>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 DBMSUTILITY.NAMEARRAY. 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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 49–7  REGISTER_USER_REPGROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>Specified replication group does not exist or the invocation database is not a master site or master materialized view site.</td>
</tr>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Incorrect privilege type was specified.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA (
    username IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the replication administrator whose privileges you want to revoke.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA (username IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the replication administrator whose privileges you want to revoke.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
</tbody>
</table>
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.
UNREGISTER_USER_REPGROUP Procedure

Parameters

Table 49–12 UNREGISTER_USER_REPGROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user you are unregistering.</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specifies the privilege type you are revoking. Use the following values for to define your privilege_type:</td>
</tr>
<tr>
<td></td>
<td>- receiver for receiver privileges</td>
</tr>
<tr>
<td></td>
<td>- proxy_snapadmin for proxy materialized view administration privileges</td>
</tr>
<tr>
<td>list_of_gnames</td>
<td>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.</td>
</tr>
<tr>
<td>table_of_gnames</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 49–13 UNREGISTER_USER_REPGROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonmaster</td>
<td>Specified replication group does not exist or the invocation database is not a master site or master materialized view site.</td>
</tr>
<tr>
<td>ORA-01917</td>
<td>User does not exist.</td>
</tr>
<tr>
<td>typefailure</td>
<td>Incorrect privilege type was specified.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“DROP_SITE_INSTANTIATION Procedure” on page 50-3</td>
<td>Public procedure that removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.</td>
</tr>
<tr>
<td>&quot;INSTANTIATE_OFFLINE Function&quot; on page 50-3</td>
<td>Public function that generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.</td>
</tr>
<tr>
<td>&quot;INSTANTIATE_ONLINE Function&quot; on page 50-6</td>
<td>Public function that generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
    refresh_template_name IN VARCHAR2,
    site_name IN VARCHAR2);
```

**Table 50–2  DROP_SITE_INSTANTIATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The name of the deployment template to be dropped.</td>
</tr>
<tr>
<td>site_name</td>
<td>Identifies the master site where you want to drop the specified template instantiation.</td>
</tr>
</tbody>
</table>

**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**

```sql
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 50–3 INSTANTIATE_OFFLINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The name of the deployment template to be instantiated.</td>
</tr>
<tr>
<td>site_name</td>
<td>The name of the remote site that is instantiating the deployment template.</td>
</tr>
<tr>
<td>runtime_parm_id</td>
<td>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).</td>
</tr>
<tr>
<td>next_date</td>
<td>The next refresh date value to be used when creating the refresh group.</td>
</tr>
<tr>
<td>interval</td>
<td>The refresh interval to be used when creating the refresh group.</td>
</tr>
<tr>
<td>use_default_gowner</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 50–4 INSTANTIATE_OFFLINE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_template_site</td>
<td>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.</td>
</tr>
<tr>
<td>not_authorized</td>
<td>The user attempting to instantiate the deployment template is not authorized to do so.</td>
</tr>
</tbody>
</table>

Returns

Table 50–5 INSTANTIATE_OFFLINE Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>
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;
```
Summary of DBMS_REPCAT_INSTANTIATE Subprograms

Table 50–6 INSTANTIATE_ONLINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The name of the deployment template to be instantiated.</td>
</tr>
<tr>
<td>site_name</td>
<td>The name of the remote site that is instantiating the deployment template.</td>
</tr>
<tr>
<td>runtime_parm_id</td>
<td>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).</td>
</tr>
<tr>
<td>next_date</td>
<td>Specifies the next refresh date value to be used when creating the refresh group.</td>
</tr>
<tr>
<td>interval</td>
<td>Specifies the refresh interval to be used when creating the refresh group.</td>
</tr>
<tr>
<td>use_default_gowner</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 50–7 INSTANTIATE_ONLINE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_template_site</td>
<td>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.</td>
</tr>
<tr>
<td>not_authorized</td>
<td>The user attempting to instantiate the deployment template is not authorized to do so.</td>
</tr>
</tbody>
</table>

Returns

Table 50–8 INSTANTIATE_ONLINE Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>
DBMS_REPCAT_RGT controls the maintenance and definition of refresh group templates.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT_RGT Subprograms
### Summary of DBMS_REPCAT_RGT Subprograms

#### Table 51–1  DBMS_REPCAT_RGT Package Subprograms  (Page 1 of 3)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ALTER_REFRESH_TEMPLATE Procedure&quot; on page 51-5</td>
<td>Allows the DBA to alter existing deployment templates.</td>
</tr>
<tr>
<td>&quot;ALTER_TEMPLATE_OBJECT Procedure&quot; on page 51-7</td>
<td>Alters objects that have been added to a specified deployment template.</td>
</tr>
<tr>
<td>&quot;ALTER_TEMPLATE_PARM Procedure&quot; on page 51-10</td>
<td>Allows the DBA to alter the parameters for a specific deployment template.</td>
</tr>
<tr>
<td>&quot;ALTER_USER_AUTHORIZATION Procedure&quot; on page 51-12</td>
<td>Alters the contents of the DBA_REPCAT_USER_AUTHORIZATIONS view.</td>
</tr>
<tr>
<td>&quot;ALTER_USER_PARM_VALUE Procedure&quot; on page 51-14</td>
<td>Changes existing parameter values that have been defined for a specific user.</td>
</tr>
<tr>
<td>&quot;COMPARE_TEMPLATES Function&quot; on page 51-16</td>
<td>Allows the DBA to compare the contents of two deployment templates.</td>
</tr>
<tr>
<td>&quot;COPY TEMPLATE Function&quot; on page 51-18</td>
<td>Allows the DBA to copy a deployment template.</td>
</tr>
<tr>
<td>&quot;CREATE_OBJECT_FROM_EXISTING Function&quot; on page 51-20</td>
<td>Creates a template object definition from existing database objects and adds it to a target deployment template.</td>
</tr>
<tr>
<td>&quot;CREATE_REFRESH TEMPLATE Function&quot; on page 51-22</td>
<td>Creates the deployment template, which allows the DBA to define the template name, private/public status, and target refresh group.</td>
</tr>
<tr>
<td>&quot;CREATE_TEMPLATE_OBJECT Function&quot; on page 51-24</td>
<td>Adds object definitions to a target deployment template container.</td>
</tr>
<tr>
<td>&quot;CREATE_TEMPLATE_PARM Function&quot; on page 51-27</td>
<td>Creates parameters for a specific deployment template to allow custom data sets to be created at the remote materialized view site.</td>
</tr>
<tr>
<td>&quot;CREATE_USER AUTHORIZATION Function&quot; on page 51-30</td>
<td>Authorizes specific users to instantiate private deployment templates.</td>
</tr>
<tr>
<td>&quot;CREATE_USER_PARM_VALUE Function&quot; on page 51-31</td>
<td>Predefines deployment template parameter values for specific users.</td>
</tr>
</tbody>
</table>
Table 51–1  DBMS_REPCAT_RGT Package Subprograms  (Page 2 of 3)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DELETE_RUNTIME_PARM procedures&quot; on page 51-33</td>
<td>Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARM procedures.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_OBJECTS Procedure&quot; on page 51-34</td>
<td>Allows the DBA to drop all objects or specific object types from a deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_TEMPLATE_PARMS Procedure&quot; on page 51-36</td>
<td>Allows the DBA to drop template parameters for a specified deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_TEMPLATE_SITES Procedure&quot; on page 51-37</td>
<td>Removes all entries from the DBA_REPCAT_TEMPLATE_SITES view.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_TEMPLATES Procedure&quot; on page 51-38</td>
<td>Removes all deployment templates at the site where the procedure is called.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_USER_AUTHORIZATIONS Procedure&quot; on page 51-38</td>
<td>Allows the DBA to drop all user authorizations for a specified deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_ALL_USER_PARM_VALUES Procedure&quot; on page 51-39</td>
<td>Drops user parameter values for a specific deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_REFRESH_TEMPLATE Procedure&quot; on page 51-40</td>
<td>Drops a deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_SITE_INSTANTIATION Procedure&quot; on page 51-41</td>
<td>Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.</td>
</tr>
<tr>
<td>&quot;DROP_TEMPLATE_OBJECT Procedure&quot; on page 51-42</td>
<td>Removes a template object from a specific deployment template.</td>
</tr>
<tr>
<td>&quot;DROP_TEMPLATE_PARM Procedure&quot; on page 51-44</td>
<td>Removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view.</td>
</tr>
<tr>
<td>&quot;DROP_USER_AUTHORIZATION Procedure&quot; on page 51-45</td>
<td>Removes a user authorization entry from the DBA_REPCAT_USER_AUTHORIZATIONS view.</td>
</tr>
<tr>
<td>&quot;DROP_USER_PARM_VALUE Procedure&quot; on page 51-46</td>
<td>Removes a predefined user parameter value for a specific deployment template.</td>
</tr>
<tr>
<td>&quot;GET_RUNTIME_PARM_ID Function&quot; on page 51-47</td>
<td>Retrieves an identification to be used when defining a runtime parameter value.</td>
</tr>
<tr>
<td>&quot;INSERT_RUNTIME_PARMS Procedure&quot; on page 51-47</td>
<td>Defines runtime parameter values prior to instantiating a template.</td>
</tr>
</tbody>
</table>
Table 51–1  DBMS_REPCAT_RGT Package Subprograms  (Page 3 of 3)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;INSTANTIATE_OFFLINE Function&quot; on page 51-49</td>
<td>Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.</td>
</tr>
<tr>
<td>&quot;INSTANTIATE_ONLINE Function&quot; on page 51-52</td>
<td>Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.</td>
</tr>
<tr>
<td>&quot;LOCK_TEMPLATE_EXCLUSIVE Procedure&quot; on page 55</td>
<td>Prevents users from reading or instantiating the template when a deployment template is being updated or modified.</td>
</tr>
<tr>
<td>&quot;LOCK_TEMPLATE_SHARED Procedure&quot; on page 51-55</td>
<td>Makes a specified deployment template read-only.</td>
</tr>
</tbody>
</table>
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);
ALTER_REFRESH_TEMPLATE Procedure

Parameters

Table 51–2 ALTER_REFRESH_TEMPLATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The name of the deployment template that you want to alter.</td>
</tr>
<tr>
<td>new_owner</td>
<td>The name of the new deployment template owner. Do not specify a value to keep the current owner.</td>
</tr>
<tr>
<td>new_refresh_group_name</td>
<td>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.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>Use this parameter to specify a new deployment template name. Do not specify a value to keep the current deployment template name.</td>
</tr>
<tr>
<td>new_template_comment</td>
<td>New deployment template comments. Do not specify a value to keep the current template comment.</td>
</tr>
<tr>
<td>new_public_template</td>
<td>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.</td>
</tr>
<tr>
<td>new_last_modified</td>
<td>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.</td>
</tr>
<tr>
<td>new_modified_by</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–3 ALTER_REFRESH_TEMPLATE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>bad_public_template</td>
<td>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.</td>
</tr>
<tr>
<td>dupl_refresh_template</td>
<td>A template with the specified name already exists.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Deployment template name that contains the object that you want to alter.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the template object that you want to alter.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of object that you want to alter.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>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.</td>
</tr>
<tr>
<td>new_object_name</td>
<td>New name of the template object. Do not specify a value to keep the current object name.</td>
</tr>
<tr>
<td>new_object_type</td>
<td>If specified, then the new object type. Objects of the following type may be specified:</td>
</tr>
<tr>
<td></td>
<td>SNAPSHOT</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td>VIEW</td>
</tr>
<tr>
<td></td>
<td>SYNONYM</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE</td>
</tr>
<tr>
<td>new_ddl_text</td>
<td>New object DDL for specified object. Do not specify any new DDL text to keep the current object DDL.</td>
</tr>
<tr>
<td>new_master_rollback_seg</td>
<td>New master rollback segment for specified object. Do not specify a value to keep the current rollback segment.</td>
</tr>
<tr>
<td>new_flavor_id</td>
<td>This parameter is for internal use only.</td>
</tr>
<tr>
<td>Note</td>
<td>Do not set this parameter unless directed to do so by Oracle Support Services.</td>
</tr>
</tbody>
</table>
Exceptions

**Table 51–5 ALTER_TEMPLATE_OBJECT Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_flavor_id</td>
<td>If you receive this exception, contact Oracle Support Services.</td>
</tr>
<tr>
<td>bad_object_type</td>
<td>Object type is specified incorrectly. See Table 51–4 for a list of valid object types.</td>
</tr>
<tr>
<td>miss_template_object</td>
<td>Template object name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_template_object</td>
<td>New template name specified in the new_refresh_template_name parameter already exists.</td>
</tr>
</tbody>
</table>

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:

```sql
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

```sql
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 51–6  ALTER_TEMPLATE_PARM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the parameter that you want to alter.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Name of the parameter that you want to alter.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>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.</td>
</tr>
<tr>
<td>new_parameter_name</td>
<td>New name of the template parameter. Do not specify a value to keep the current parameter name.</td>
</tr>
<tr>
<td>new_default_parm_value</td>
<td>New default value for the specified parameter. Do not specify a value to keep the current default value.</td>
</tr>
<tr>
<td>new_prompt_string</td>
<td>New prompt text for the specified parameter. Do not specify a value to keep the current prompt string.</td>
</tr>
<tr>
<td>new_user_override</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–7  ALTER_TEMPLATE_PARM Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_template_parm</td>
<td>Template parameter specified is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_template_parm</td>
<td>Combination of new_refresh_template_name and new_parameter_name already exists.</td>
</tr>
</tbody>
</table>
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:

```sql
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 51–8  ALTER_USER_AUTHORIZATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>Name of the user whose authorization you want to alter.</td>
</tr>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that is currently assigned to the specified user that you want to alter.</td>
</tr>
<tr>
<td>new_user_name</td>
<td>Use this parameter to define a new user for this template authorization. Do not specify a value to keep the current user.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–9  ALTER_USER_AUTHORIZATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_user_authorization</td>
<td>The combination of user_name and refresh_template_name values specified does not exist in the DBA_REPCAT_USER_AUTHORIZATIONS view.</td>
</tr>
<tr>
<td>miss_user</td>
<td>The user name specified for the new_user_name or user_name parameter is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template specified for the new_refresh_template parameter is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_user_authorization</td>
<td>A row already exists for the specified user name and deployment template name.</td>
</tr>
</tbody>
</table>
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**

```sql
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 51–10  ALTER_USER_PARM_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the user parameter value that you want to alter.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Name of the parameter that you want to alter.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the user whose parameter value you want to alter.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>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.</td>
</tr>
<tr>
<td>new_parameter_name</td>
<td>The new template parameter name. Do not specify a value to keep the user value defined for the existing parameter.</td>
</tr>
<tr>
<td>new_user_name</td>
<td>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.</td>
</tr>
<tr>
<td>new_parm_value</td>
<td>The new parameter value for the specified user parameter. Do not specify a value to keep the current parameter value.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–11  ALTER_USER_PARM_VALUE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_template_parm</td>
<td>Template parameter specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>User name specified for the user_name or new_user_name parameters is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user_parm_values</td>
<td>User parameter value specified does not exist.</td>
</tr>
<tr>
<td>dupl_user_parm_values</td>
<td>New user parameter specified already exists.</td>
</tr>
</tbody>
</table>
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:

```sql
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;
/
```

COMPARETEMPLATES 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 COMPARETEMPLATES function returns a number that you specify in the WHERE clause when querying the USER_REPCAT_TEMP_OUTPUT temporary view. For example, if the COMPARETEMPLATES 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):

```sql
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.
Summary of DBMS_REPCAT_RGT Subprograms

Syntax

DBMS_REPCAT_RGT.COMPARE_TEMPLATES (  
    source_template_name  IN VARCHAR2,  
    compare_template_name IN VARCHAR2)  
return NUMBER;

Parameters

Table 51–12 COMPARE_TEMPLATES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_template_name</td>
<td>Name of the first deployment template to be compared.</td>
</tr>
<tr>
<td>compare_template_name</td>
<td>Name of the second deployment template to be compared.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–13 COMPARE_TEMPLATES Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name to be compared is invalid or does not exist.</td>
</tr>
</tbody>
</table>

Returns

Table 51–14 COMPARE_TEMPLATES Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>
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**

```sql
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;
```
Summary of DBMS_REPCAT_RGT Subprograms

Parameters

Table 51–15  COPY_TEMPLATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_refresh_template_name</td>
<td>Name of the deployment template to be copied.</td>
</tr>
<tr>
<td>new_refresh_template_name</td>
<td>Name of the new deployment template.</td>
</tr>
<tr>
<td>copy_user_authorizations</td>
<td>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.</td>
</tr>
<tr>
<td>dblink</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–16  COPY_TEMPLATE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name to be copied is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_refresh_template</td>
<td>Name of the new refresh template specified already exists.</td>
</tr>
<tr>
<td>bad_copy_auth</td>
<td>Value specified for the copy_user_authorizations parameter is invalid. Valid values are Y, N, and NULL.</td>
</tr>
</tbody>
</table>

Returns

Table 51–17  COPYTEMPLATES Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number &gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template to which you want to add this object.</td>
</tr>
<tr>
<td>object_name</td>
<td>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).</td>
</tr>
<tr>
<td>sname</td>
<td>The schema that contains the object that you are creating your template object from.</td>
</tr>
<tr>
<td>oname</td>
<td>Name of the object that you are creating your template object from.</td>
</tr>
<tr>
<td>otype</td>
<td>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</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of existing deployment templates.</td>
</tr>
<tr>
<td>bad_object_type</td>
<td>The object type is specified incorrectly.</td>
</tr>
<tr>
<td>dupl_template_object</td>
<td>An object of the same name and type has already been added to the specified deployment template.</td>
</tr>
<tr>
<td>objectmissing</td>
<td>The object specified does not exist.</td>
</tr>
</tbody>
</table>
CREATE_REFRESH_TEMPLATE Function

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>

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

```sql
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;
```
Summary of DBMS_REPCAT_RGT Subprograms

Parameters

Table 51–21  CREATE_REFRESH_TEMPLATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>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.</td>
</tr>
<tr>
<td>refresh_group_name</td>
<td>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.</td>
</tr>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that you are creating. This name is referenced in all activities that involve this deployment template.</td>
</tr>
<tr>
<td>template_comment</td>
<td>User comments defined with this parameter are listed in the DBA_REPCAT_REFRESH_TEMPLATES view.</td>
</tr>
<tr>
<td>public_template</td>
<td>Specifies whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private).</td>
</tr>
<tr>
<td>last_modified</td>
<td>The date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.</td>
</tr>
<tr>
<td>modified_by</td>
<td>Name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.</td>
</tr>
<tr>
<td>creation_date</td>
<td>The date that this deployment template was created. If a value is not specified, then the current date is automatically used.</td>
</tr>
<tr>
<td>created_by</td>
<td>Name of the user who created this deployment template. If a value is not specified, then the current user is automatically used.</td>
</tr>
</tbody>
</table>
CREATE_TEMPLATE_OBJECT Function

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dupl_refresh_template</td>
<td>A template with the specified name already exists.</td>
</tr>
<tr>
<td>bad_public_template</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>

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

```sql
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 51–24  CREATE_TEMPLATE_OBJECT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template to which you want to add this object.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the template object that you are creating.</td>
</tr>
<tr>
<td>object_type</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER SEQUENCE DATABASE LINK</td>
</tr>
<tr>
<td>ddl_text</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>master_rollback_seg</td>
<td>Specifies the name of the rollback segment to use when executing the defined object DDL at the remote materialized view site.</td>
</tr>
<tr>
<td>flavor_id</td>
<td>This parameter is for internal use only.</td>
</tr>
</tbody>
</table>

Note: Do not set this parameter unless directed to do so by Oracle Support Services.
CREATE TEMPLATE OBJECT Function

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of existing deployment templates.</td>
</tr>
<tr>
<td>bad_object_type</td>
<td>Object type is specified incorrectly. See Table 51–24 for a list of valid object types.</td>
</tr>
<tr>
<td>dupl_template_object</td>
<td>An object of the same name and type has already been added to the specified deployment template.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>
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:

```sql
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 and/or prompt strings (see "ALTER_TEMPLATE_PARM Procedure" on page 51-10 for more information). The number returned by this function is used internally by Oracle to manage deployment templates.
CREATE_TEMPLATE_PARM Function

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template for which you want to create the parameter.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Name of the parameter you are creating.</td>
</tr>
<tr>
<td>default_parm_value</td>
<td>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.</td>
</tr>
<tr>
<td>prompt_string</td>
<td>The descriptive prompt text that is displayed for this template parameter during the instantiation process.</td>
</tr>
<tr>
<td>user_override</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The specified refresh template name is invalid or missing.</td>
</tr>
<tr>
<td>dupl_template_parm</td>
<td>A parameter of the same name has already been defined for the specified deployment template.</td>
</tr>
</tbody>
</table>
Returns

Table 51–29 CREATE_TEMPLATE_PARM Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>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')</td>
</tr>
<tr>
<td>refresh_template_name</td>
<td>Name of the template that you want to authorize the specified user to instantiate.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_user</td>
<td>User name supplied is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_refresh_template</td>
<td>Refresh template name supplied is invalid or does not exist.</td>
</tr>
<tr>
<td>dupl_user_authorization</td>
<td>An authorization has already been created for the specified user and deployment template.</td>
</tr>
</tbody>
</table>
Returns

Table 51–32 CREATE_USER_AUTHORIZATION Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>

**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**

```sql
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 51–33 CREATE_USER_PARM_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Specifies the name of the deployment template that contains the parameter you are creating a user parameter value for.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Name of the template parameter that you are defining a user parameter value for.</td>
</tr>
<tr>
<td>user_name</td>
<td>Specifies the name of the user that you are predefining a user parameter value for.</td>
</tr>
<tr>
<td>parm_value</td>
<td>The predefined parameter value that will be used during the instantiation process initiated by the specified user.</td>
</tr>
</tbody>
</table>
CREATE_USER_PARM_VALUE Function

Exceptions

Table 51–34  CREATE_USER_PARM_VALUE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or missing.</td>
</tr>
<tr>
<td>dupl_user_parm_values</td>
<td>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.</td>
</tr>
<tr>
<td>miss_template_parm</td>
<td>Specified deployment template parameter name is invalid or missing.</td>
</tr>
<tr>
<td>miss_user</td>
<td>Specified user name is invalid or missing.</td>
</tr>
</tbody>
</table>

Returns

Table 51–35  CREATE_USER_PARM_VALUE Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>System-generated number used internally by Oracle.</td>
</tr>
</tbody>
</table>
Summary of DBMS_REPCAT_RGT Subprograms

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_P ARMS Procedure

Use this procedure before instantiating a deployment template to delete a runtime parameter value that you defined using the INSERT_RUNTIME_P ARMS procedure.

Syntax

DBMS_REPCAT_RGT.DELETE_RUNTIME_P ARMS(
    runtime_parm_id IN NUMBER,
    parameter_name IN VARCHAR2);
DROP_ALL_OBJECTS Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime_parm_id</td>
<td>Specifies the identification that you previously assigned the runtime parameter value to (this value was retrieved using the GET_RUNTIME_PARM_ID function).</td>
</tr>
<tr>
<td>parameter_name</td>
<td>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).</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_template_parm</td>
<td>The specified deployment template parameter name is invalid or missing.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT_RGT.DROP_ALL_OBJECTS (
    refresh_template_name IN VARCHAR2,
    object_type IN VARCHAR2 := NULL);
```
Parameters

Table 51–38  DROP_ALL_OBJECTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the objects that you want to drop.</td>
</tr>
<tr>
<td>object_type</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>SNAPSHOT</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td>VIEW</td>
</tr>
<tr>
<td></td>
<td>SYNONYM</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–39  DROP_ALL_OBJECTS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
<tr>
<td>bad_object_type</td>
<td>Object type is specified incorrectly. See Table 51–38 for a list of valid object types.</td>
</tr>
</tbody>
</table>
DROP_ALL_TEMPLATE_P ARMS 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the parameters and objects that you want to drop.</td>
</tr>
<tr>
<td>drop_objects</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the sites that you want to drop.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATES;
```

**Parameters**

None

**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**

```sql
DBMS_REPCAT_RGT.DROP_ALL_USER_AUTHORIZATIONS (   
    refresh_template_name   IN   VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the user authorizations that you want to drop.</td>
</tr>
</tbody>
</table>
 Exceptions

Table 51–45  DROP_ALL_USER_AUTHORIZATIONS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
</tbody>
</table>

 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:

- `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
- `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);  
```
DROP_REFRESH_TEMPLATE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template that contains the parameter values that you want to drop.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the user whose parameter values you want to drop.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Template parameter that contains the values that you want to drop.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>User name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user_parm_values</td>
<td>Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT_RGT.DROP_REFRESH_TEMPLATE (  
    refresh_template_name IN VARCHAR2);
```
Summary of DBMS_REPCAT_RGT Subprograms

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template to be dropped.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of deployment templates.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The name of the deployment template to be dropped.</td>
</tr>
<tr>
<td>user_name</td>
<td>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.</td>
</tr>
<tr>
<td>site_name</td>
<td>Identifies the master site where you want to drop the specified template instantiation.</td>
</tr>
</tbody>
</table>
DROP_TEMPLATE_OBJECT Procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>The username specified does not exist.</td>
</tr>
<tr>
<td>miss_template_site</td>
<td>The deployment template has not been instantiated for user and site.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPCAT_RGT.DROP_TEMPLATE_OBJECT (
    refresh_template_name IN VARCHAR2,
    object_name IN VARCHAR2,
    object_type IN VARCHAR2);
```
### Parameters

**Table 51–52 DROP TEMPLATE OBJECT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template from which you are dropping the object.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the template object to be dropped.</td>
</tr>
<tr>
<td>object_type</td>
<td>The type of object that is to be dropped. Objects of the following type may be specified:</td>
</tr>
<tr>
<td></td>
<td>SNAPSHOT PROCEDURE</td>
</tr>
<tr>
<td></td>
<td>INDEX FUNCTION</td>
</tr>
<tr>
<td></td>
<td>TABLE PACKAGE</td>
</tr>
<tr>
<td></td>
<td>VIEW PACKAGE BODY</td>
</tr>
<tr>
<td></td>
<td>SYNONYM TRIGGER</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE DATABASE LINK</td>
</tr>
</tbody>
</table>

### Exceptions

**Table 51–53 DROP TEMPLATE OBJECT Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_template_object</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_REPCAT_RGT.DROP_TEMPLATE_PARM (  
    refresh_template_name IN VARCHAR2,  
    parameter_name IN VARCHAR2);
```

Parameters

**Table 51–54  DROP_TEMPLATE_PARM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>The deployment template name that has the parameter that you want to drop</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Name of the parameter that you want to drop</td>
</tr>
</tbody>
</table>

Exceptions

**Table 51–55  DROP_TEMPLATE_PARM Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_template_parm</td>
<td>The parameter name specified is invalid or does not exist. Query the DBA_REPCAT TEMPLATE_PARMS view to see a list of template parameters.</td>
</tr>
</tbody>
</table>
**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 51-38

**Syntax**

```sql
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
    refresh_template_name IN VARCHAR2,
    user_name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template from which the user’s authorization is being removed.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the user whose authorization is being removed.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_user</td>
<td>Specified user name is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user_authorization</td>
<td>Specified user and deployment template combination does not exist. Query the <code>DBA_REPCAT_USER_AUTHORIZATIONS</code> view to see a list of user/deployment template authorizations.</td>
</tr>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Deployment template name that contains the parameter value that you want to drop.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Parameter name that contains the predefined value that you want to drop.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the user whose parameter value you want to drop.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>User name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user_parm_values</td>
<td>Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.</td>
</tr>
</tbody>
</table>
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;
```

Parameters

None

Returns

Table 51–60 GET_RUNTIME_PARM_ID Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Corresponding Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>Runtime parameter values are assigned to the system-generated number and are also used during the instantiation process.</td>
</tr>
</tbody>
</table>

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 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); 
```
INSERT_RUNTIME_PARMS Procedure

Parameters

Table 51–61  INSERT_RUNTIME_PARMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime_parm_id</td>
<td>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.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>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.</td>
</tr>
<tr>
<td>parameter_value</td>
<td>The runtime parameter value that you want to use during the deployment template instantiation process.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–62  INSERT_RUNTIME_PARMS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>The deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>The user name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user_parm_values</td>
<td>The deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.</td>
</tr>
</tbody>
</table>
Usage Notes

Because 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:

```sql
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 is 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.
**INSTANTIATE_OFFLINE Function**

**Syntax**

```sql
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;
```

**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 51-49 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.
Parameters

Table 51–63 INSTANTIATE_OFFLINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template to be instantiated.</td>
</tr>
<tr>
<td>site_name</td>
<td>Name of the remote site that is instantiating the deployment template.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the authorized user who is instantiating the deployment template.</td>
</tr>
<tr>
<td>runtime_parm_id</td>
<td>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).</td>
</tr>
<tr>
<td>next_date</td>
<td>Specifies the next refresh date value to be used when creating the refresh group.</td>
</tr>
<tr>
<td>interval</td>
<td>Specifies the refresh interval to be used when creating the refresh group.</td>
</tr>
<tr>
<td>use_default_gowner</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

Table 51–64 INSTANTIATE_OFFLINE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Deployment template name specified is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>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.</td>
</tr>
</tbody>
</table>
INSTANTIATE_ONLINE Function

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;system-generated number&gt;</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

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 51-49 section.
Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_template_name</td>
<td>Name of the deployment template to be instantiated.</td>
</tr>
<tr>
<td>site_name</td>
<td>Name of the remote site that is instantiating the deployment template.</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the authorized user who is instantiating the deployment template.</td>
</tr>
<tr>
<td>runtime_parm_id</td>
<td>If you have defined runtime parameter values using the <code>INSERT_RUNTIME_PARMS</code> procedure, then specify the identification used when creating the runtime parameters (the identification was retrieved by using the <code>GET_RUNTIME_PARM_ID</code> function).</td>
</tr>
<tr>
<td>next_date</td>
<td>Specifies the next refresh date value to be used when creating the refresh group.</td>
</tr>
<tr>
<td>interval</td>
<td>Specifies the refresh interval to be used when creating the refresh group.</td>
</tr>
<tr>
<td>use_default_gowner</td>
<td>If true, then any materialized view groups created are owned by the default user <code>PUBLIC</code>. If false, then any materialized view groups created are owned by the user performing the instantiation.</td>
</tr>
</tbody>
</table>
INSTANTIATE_ONLINE Function

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_refresh_template</td>
<td>Specified deployment template name is invalid or does not exist.</td>
</tr>
<tr>
<td>miss_user</td>
<td>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.</td>
</tr>
<tr>
<td>bad_parms</td>
<td>Not all of the template parameters were populated by the defined user parameter values and/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).</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system-generated number&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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**

```sql
DBMS_REPCAT_RGT.LOCK_TEMPLATE_EXCLUSIVE();
```

**Parameters**

None

**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**

```sql
DBMS_REPCAT_RGT.LOCK_TEMPLATE_SHARED();
```

**Parameters**

None
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;REPLICATION_OFF Procedure&quot; on page 52-3</td>
<td>Modifies tables without replicating the modifications to any other sites in the replication environment, or disables row-level replication when using procedural replication.</td>
</tr>
<tr>
<td>&quot;REPLICATION_ON Procedure&quot; on page 52-3</td>
<td>Re-enables replication of changes after replication has been temporarily suspended.</td>
</tr>
<tr>
<td>&quot;REPLICATION_IS_ON Function&quot; on page 52-4</td>
<td>Determines whether or not replication is running.</td>
</tr>
<tr>
<td>&quot;FROM_REMOTE Function&quot; on page 52-4</td>
<td>Returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures.</td>
</tr>
<tr>
<td>&quot;GLOBAL_NAME Function&quot; on page 52-5</td>
<td>Determines the global database name of the local database (the global name is the returned value).</td>
</tr>
</tbody>
</table>
| "MAKE_INTERNAL_PKG Procedure" on page 52-5 | 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 52-6 | 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();

Parameters

None

REPLICATION_ON Procedure

This procedure re-enables replication of changes after replication has been temporarily suspended.

Syntax

DBMS_REPUTIL.REPLICATION_ON();

Parameters

None
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

```sql
DBMS_REPUTIL.REPLICATION_IS_ON()
return BOOLEAN;
```

Parameters

None

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

```sql
DBMS_REPUTIL.FROM_REMOTE()
return BOOLEAN;
```

Parameters

None
GLOBAL_NAME Function

This function determines the global database name of the local database (the global name is the returned value).

**Syntax**

```sql
DBMS_REPUTIL.GLOBAL_NAME()
return VARCHAR2;
```

**Parameters**

None

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**

```sql
DBMS_REPUTIL.MAKE_INTERNAL_PKG (  
    canon_sname IN VARCHAR2,  
    canon_oname IN VARCHAR2);
```
SYNC_UP_REP Procedure

Parameters

Table 52–2 MAKE_INTERNAL_PKG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canon_sname</td>
<td>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).</td>
</tr>
<tr>
<td>canon_oname</td>
<td>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).</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_REPUTIL.SYNC_UP_REP (  
  canon_sname IN VARCHAR2,  
  canon_oname IN VARCHAR2);  
```
Parameters

Table 52–3  SYNC_UP_REP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canon_sname</td>
<td>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).</td>
</tr>
<tr>
<td>canon_oname</td>
<td>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).</td>
</tr>
</tbody>
</table>
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:

- Summary of `DBMS_LOB` Subprograms
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 DBMSRESOURCE_MANAGER Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CREATE_PLAN Procedure&quot; on page 53-3</td>
<td>Creates entries which define resource plans.</td>
</tr>
<tr>
<td>&quot;CREATE_SIMPLE_PLAN Procedure&quot; on page 53-4</td>
<td>Creates a single-level resource plan containing up to eight consumer groups in one step.</td>
</tr>
<tr>
<td>&quot;UPDATE_PLAN Procedure&quot; on page 53-5</td>
<td>Updates entries which define resource plans.</td>
</tr>
<tr>
<td>&quot;DELETE_PLAN Procedure&quot; on page 53-5</td>
<td>Deletes the specified plan as well as all the plan directives it refers to.</td>
</tr>
<tr>
<td>&quot;DELETE_PLAN_CASCADE Procedure&quot; on page 53-6</td>
<td>Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups).</td>
</tr>
<tr>
<td>&quot;CREATE_CONSUMER_GROUP Procedure&quot; on page 53-7</td>
<td>Creates entries which define resource consumer groups.</td>
</tr>
<tr>
<td>&quot;UPDATE_CONSUMER_GROUP Procedure&quot; on page 53-7</td>
<td>Updates entries which define resource consumer groups.</td>
</tr>
<tr>
<td>&quot;DELETE_CONSUMER_GROUP Procedure&quot; on page 53-8</td>
<td>Deletes entries which define resource consumer groups.</td>
</tr>
<tr>
<td>&quot;CREATE_PLAN_DIRECTIVE Procedure&quot; on page 53-8</td>
<td>Creates resource plan directives.</td>
</tr>
<tr>
<td>&quot;UPDATE_PLAN_DIRECTIVE Procedure&quot; on page 53-10</td>
<td>Updates resource plan directives.</td>
</tr>
<tr>
<td>&quot;DELETE_PLAN_DIRECTIVE Procedure&quot; on page 53-12</td>
<td>Deletes resource plan directives.</td>
</tr>
<tr>
<td>&quot;CREATE_PENDING_AREA Procedure&quot; on page 53-12</td>
<td>Creates a work area for changes to resource manager objects.</td>
</tr>
</tbody>
</table>
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan.</td>
</tr>
<tr>
<td>comment</td>
<td>User’s comment.</td>
</tr>
<tr>
<td>cpu_mth</td>
<td>Allocation method for CPU resources.</td>
</tr>
<tr>
<td>active_sess_pool_mth</td>
<td>Allocation method for maximum active sessions.</td>
</tr>
<tr>
<td>parallel_degree_limit_mth</td>
<td>Allocation method for degree of parallelism.</td>
</tr>
<tr>
<td>new_queueing_mth</td>
<td>Specifies type of queuing policy to use with active session pool feature.</td>
</tr>
</tbody>
</table>

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan.</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user’s comment.</td>
</tr>
<tr>
<td>new_cpu_mth</td>
<td>Name of new allocation method for CPU resources.</td>
</tr>
<tr>
<td>new_active_sess_pool_mth</td>
<td>Name of new method for maximum active sessions.</td>
</tr>
<tr>
<td>new_parallel_degree_limit_mth</td>
<td>Name of new method for degree of parallelism.</td>
</tr>
<tr>
<td>new_queueing_mth</td>
<td>Specifies type of queuing policy to use with active session pool feature.</td>
</tr>
</tbody>
</table>

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.
DELETE_PLAN_CASCADE Procedure

Syntax

```sql
DBMS_RESOURCE_MANAGER.DELETE_PLAN (plan IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan to delete.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (plan IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of plan.</td>
</tr>
</tbody>
</table>

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 to specify it when creating or updating a plan.

Usage Notes

Defaults are:

53-6  Supplied PL/SQL Packages and Types Reference
**Summary of DBMS_RESOURCE_MANAGER Subprograms**

- `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**

```sql
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP ( 
  consumer_group IN VARCHAR2,
  comment IN VARCHAR2,
  cpu_mth IN VARCHAR2 DEFAULT 'ROUND-ROBIN');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
<tr>
<td>comment</td>
<td>User’s comment.</td>
</tr>
<tr>
<td>cpu_mth</td>
<td>Name of CPU resource allocation method.</td>
</tr>
</tbody>
</table>

### UPDATE_CONSUMER_GROUP Procedure

This procedure lets you update entries which define resource consumer groups.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP ( 
  consumer_group IN VARCHAR2,
  new_comment IN VARCHAR2 DEFAULT NULL,
  new_cpu_mth IN VARCHAR2 DEFAULT NULL);
```
DELETE_CONSUMER_GROUP Procedure

Parameters

Table 53–7  UPDATE_CONSUMER_GROUP Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user’s comment.</td>
</tr>
<tr>
<td>new_cpu_mth</td>
<td>Name of new method for CPU resource allocation.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (  
    consumer_group IN VARCHAR2);
```

Parameters

Table 53–8  DELETE_CONSUMER_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of consumer group to be deleted.</td>
</tr>
</tbody>
</table>

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

```sql
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,  
```
Summary of DBMS_RESOURCE_MANAGER Subprograms

Parameters

Table 53–9  CREATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan.</td>
</tr>
<tr>
<td>group_or_subplan</td>
<td>Name of consumer group or subplan.</td>
</tr>
<tr>
<td>comment</td>
<td>Comment for the plan directive.</td>
</tr>
<tr>
<td>cpu_p1</td>
<td>First parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p2</td>
<td>Second parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p3</td>
<td>Third parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p4</td>
<td>Fourth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p5</td>
<td>Fifth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p6</td>
<td>Sixth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p7</td>
<td>Seventh parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>cpu_p8</td>
<td>Eighth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>active_sess_pool_pl</td>
<td>First parameter for the maximum active sessions allocation method (Reserved for future use).</td>
</tr>
<tr>
<td>queueing_pl</td>
<td>queue timeout in seconds</td>
</tr>
<tr>
<td>switch_group</td>
<td>group to switch into once switch time is reached</td>
</tr>
<tr>
<td>switch_time</td>
<td>switch time</td>
</tr>
</tbody>
</table>

```sql
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_pl IN NUMBER DEFAULT UNLIMITED,
queueing_pl 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_pl IN NUMBER DEFAULT UNLIMITED);
```
UPDATE_PLAN_DIRECTIVE Procedure

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

```sql
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_pl IN NUMBER DEFAULT NULL,
    new_parallel_degree_limit_pl 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan.</td>
</tr>
<tr>
<td>group_or_subplan</td>
<td>Name of consumer group or subplan.</td>
</tr>
<tr>
<td>comment</td>
<td>Comment for the plan directive.</td>
</tr>
<tr>
<td>new_cpu_p1</td>
<td>First parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p2</td>
<td>Second parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p3</td>
<td>Third parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p4</td>
<td>Fourth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p5</td>
<td>Fifth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p6</td>
<td>Sixth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p7</td>
<td>Seventh parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_cpu_p8</td>
<td>Eighth parameter for the CPU resource allocation method.</td>
</tr>
<tr>
<td>new_active_sess_pool_p1</td>
<td>First parameter for the maximum active sessions allocation method (Reserved for future use).</td>
</tr>
<tr>
<td>new_queueing_p1</td>
<td>queue timeout in seconds</td>
</tr>
<tr>
<td>new_switch_group</td>
<td>group to switch into once switch time is reached</td>
</tr>
<tr>
<td>new_switch_time</td>
<td>switch time</td>
</tr>
<tr>
<td>new_switch_estimate</td>
<td>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.</td>
</tr>
<tr>
<td>new_max_est_exec_time</td>
<td>maximum estimated execution time in seconds</td>
</tr>
<tr>
<td>new_undo_pool</td>
<td>undo pool size for the consumer group, in Kbytes</td>
</tr>
<tr>
<td>new_parallel_degree_limit_p1</td>
<td>First parameter for the degree of parallelism allocation method.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 53–11 DELETE_PLAN_DIRECTIVE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>plan</td>
</tr>
<tr>
<td>group_or_subplan</td>
</tr>
</tbody>
</table>

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 have 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 Oracle8i, the plan directive parameter, `parallel_degree_limit_pl`, may only appear in plan directives that refer to consumer groups (i.e., not at subplans).
7. There cannot be more than 32 plan directives coming from any given plan (i.e., 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 above 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.
CLEAR_PENDING_AREA Procedure

Syntax

```sql
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

CLEAR_PENDING_AREA Procedure

This procedure lets you clear pending changes for the resource manager.

Syntax

```sql
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

```sql
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>
<thead>
<tr>
<th>Subplan/Group</th>
<th>CPU_Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAILDB Plan</td>
<td>30%</td>
</tr>
<tr>
<td>BUGDB Plan</td>
<td>70%</td>
</tr>
</tbody>
</table>

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. The following diagram depicts this scenario:

![Diagram](image-url)

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 multi-plan (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 multi-plan definition in the above 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
```
SUBMIT_PENDING_AREA Procedure

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_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);}

DBMSRESOURCE_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);}

DBMSRESOURCE_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);}
'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 above 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 53–13  SET_INITIAL_CONSUMER_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Name of the user.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>The user’s initial consumer group.</td>
</tr>
</tbody>
</table>

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.
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

```sql
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (  
    session_id IN NUMBER,  
    session_serial IN NUMBER,  
    consumer_group IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>SID column from the view V$SESSION.</td>
</tr>
<tr>
<td>session_serial</td>
<td>SERIAL# column from view V$SESSION.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group to switch to.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (user IN VARCHAR2,
consumer_group IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Name of the user.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group to switch to.</td>
</tr>
</tbody>
</table>

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.
The DBMSResourceManager_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 DBMSResourceManager_PRIVS Subprograms
Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms

Table 54–1 DBMS_RESOURCE_MANAGER_PRIVS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GRANT_SYSTEM_PRIVILEGE Procedure&quot; on page 54-2</td>
<td>Performs a grant of a system privilege.</td>
</tr>
<tr>
<td>&quot;REVOKE_SYSTEM_PRIVILEGE Procedure&quot; on page 54-3</td>
<td>Performs a revoke of a system privilege.</td>
</tr>
<tr>
<td>&quot;GRANT SWITCH_CONSUMER_GROUP Procedure&quot; on page 54-3</td>
<td>Grants the privilege to switch to resource consumer groups.</td>
</tr>
<tr>
<td>&quot;REVOKE SWITCH_CONSUMER_GROUP Procedure&quot; on page 54-5</td>
<td>Revokes the privilege to switch to resource consumer groups.</td>
</tr>
</tbody>
</table>

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 54–2 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee_name</td>
<td>Name of the user or role to whom privilege is to be granted.</td>
</tr>
<tr>
<td>privilege_name</td>
<td>Name of the privilege to be granted.</td>
</tr>
<tr>
<td>admin_option</td>
<td>TRUE if the grant is with admin_option, FALSE otherwise.</td>
</tr>
</tbody>
</table>

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:

```sql
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

```sql
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (  
    revokee_name IN VARCHAR2,  
    privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revokee_name</td>
<td>Name of the user or role from whom privilege is to be revoked.</td>
</tr>
<tr>
<td>privilege_name</td>
<td>Name of the privilege to be revoked.</td>
</tr>
</tbody>
</table>

Example

The following call revokes the ADMINISTER_RESOURCE_MANAGER from user scott:

```sql
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

```sql
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (  
    grantee_name IN VARCHAR2,
```
Grant Switch Consumer Group Procedure

consumer_group IN VARCHAR2,
grant_option IN BOOLEAN);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee_name</td>
<td>Name of the user or role to whom privilege is to be granted.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
<tr>
<td>grant_option</td>
<td>TRUE if grantee should be allowed to grant access, FALSE otherwise.</td>
</tr>
</tbody>
</table>

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 53, "DBMS_RESOURCE_MANAGER"

Example

DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_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

```sql
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
    revokee_name IN VARCHAR2,
    consumer_group IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revokee_name</td>
<td>Name of user/role from which to revoke access.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
</tbody>
</table>

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 via that role will not be subsequently 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 via PUBLIC will not be subsequently able to switch to that consumer group.

Example

The following example revokes the privileges to switch to mail_maintenance_group from Scott:

```sql
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (  
    'scott', 'mail_maintenance_group');
```
REVOKE_SWITCH_CONSUMER_GROUP Procedure
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 a statement is suspended, the act of suspending should be logged 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 `DBMS_RESUMABLE.ABORT()` procedure.

This chapter discusses the following topics:

- **Summary of DBMS_RESUMABLE Subprograms**
Summary of DBMS_RESUMABLE Subprograms

Table 55–1  DBMS_RESUMABLE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ABORT Procedure&quot; on page 55-2</td>
<td>Aborts a suspended resumable space allocation.</td>
</tr>
<tr>
<td>&quot;GET_SESSION_TIMEOUT Function&quot; on page 55-3</td>
<td>Returns the current timeout value of the resumable space allocations for a session with session_id.</td>
</tr>
<tr>
<td>&quot;SET_SESSION_TIMEOUT Procedure&quot; on page 55-3</td>
<td>Sets the timeout of resumable space allocations for a session with session_id.</td>
</tr>
<tr>
<td>&quot;GET_TIMEOUT Function&quot; on page 55-4</td>
<td>Returns the current timeout value of resumable space allocations for the current session.</td>
</tr>
<tr>
<td>&quot;SET_TIMEOUT Procedure&quot; on page 55-4</td>
<td>Sets the timeout of resumable space allocations for the current session.</td>
</tr>
<tr>
<td>&quot;SPACE_ERROR_INFO Function&quot; on page 55-4</td>
<td>Looks for space-related errors in the error stack. If it cannot find a space-related error, it will return FALSE.</td>
</tr>
</tbody>
</table>

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 55–2 shows the parameters for the ABORT procedure.
**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**

```sql
DBMS_RESUMABLE.GET_SESSION_TIMEOUT (
    session_id  IN NUMBER);
```

**Parameters**

Table 55–3 shows the parameters for the `GET_SESSION_TIMEOUT` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>The session identifier of the resumable space allocation.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_RESUMABLE.SET_SESSION_TIMEOUT (
    session_id   IN NUMBER,
    timeout      IN NUMBER);
```

**Parameters**

Table 55–4 shows the parameters for the `SET_SESSION_TIMEOUT` procedure.
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 55-5 shows the parameters for the SET_TIMEOUT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout of the resumable space allocation.</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_RESUMABLE Subprograms

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_type</td>
<td>The space error type. It will be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- NO MORE SPACE</td>
</tr>
<tr>
<td></td>
<td>- MAX EXTENTS REACHED</td>
</tr>
<tr>
<td></td>
<td>- SPACE QUOTA EXCEEDED</td>
</tr>
<tr>
<td>object_type</td>
<td>The object type. It will be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- TABLE SPACE</td>
</tr>
<tr>
<td></td>
<td>- ROLLBACK SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- UNDO SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- TABLE</td>
</tr>
<tr>
<td></td>
<td>- INDEX</td>
</tr>
<tr>
<td></td>
<td>- CLUSTER</td>
</tr>
<tr>
<td></td>
<td>- TEMP SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- INDEX PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE PARTITION</td>
</tr>
<tr>
<td></td>
<td>- LOB SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- TABLE SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- LOB SUBPARTITION</td>
</tr>
<tr>
<td>object_owner</td>
<td>The owner of the object. NULL if it cannot be determined.</td>
</tr>
<tr>
<td>table_space_name</td>
<td>The table space where the object resides. NULL if it cannot be determined.</td>
</tr>
</tbody>
</table>
### Table 55–6  SPACE_ERROR_INFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of rollback segment, temp segment, table, index, or cluster.</td>
</tr>
<tr>
<td>sub_object_name</td>
<td>The partition name or sub-partition name of LOB, TABLE, or INDEX. NULL if it cannot be determined.</td>
</tr>
</tbody>
</table>
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 fuller discussion and more usage information on DBMS_RLS.

This chapter discusses the following topics:

- Dynamic Predicates
- Security
- Usage Notes
- Summary of DBMS_RLS Subprograms
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 or view is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```plsql
DBMS_RLS.ADD_POLICY ('scott', 'emp', 'emp_policy', 'secusr', 'emp_sec', 'select');
```

Whenever the EMP table, under SCOTT schema, is referenced in a query or subquery (SELECT), the server calls the EMP_SEC function (under SECUSR schema). This returns a predicate specific to the current user for the EMP_POLICY policy. The policy function may generate the predicates based on whatever session environment variables are 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:

```sql
SELECT * FROM scott.emp WHERE P1
```

Here, P1 (e.g., SAL > 10000, or even a subquery) is the predicate returned from the EMP_SEC function. The server treats the EMP 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 also 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 and enable/disable security policies. For example, you can drop or disable the EMP_POLICY with the following PL/SQL statements:
Summary of DBMS_RLS Subprograms

DBMS_RLS.DROP_POLICY('scott', 'emp', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('scott', 'emp', 'emp_policy', FALSE)

Security

A security check is performed when the transient view is created with subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view’s definer for the purpose of security check and object look-up.

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.

Summary of DBMS_RLS Subprograms

<table>
<thead>
<tr>
<th>Table 56–1  DBMS_RLS Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subprogram</strong></td>
</tr>
<tr>
<td>&quot;ADD_POLICY Procedure&quot; on page 56-4</td>
</tr>
<tr>
<td>&quot;DROP_POLICY Procedure&quot; on page 56-6</td>
</tr>
<tr>
<td>&quot;REFRESH_POLICY Procedure&quot; on page 56-7</td>
</tr>
<tr>
<td>&quot;ENABLE_POLICY Procedure&quot; on page 56-7</td>
</tr>
<tr>
<td>&quot;CREATE_POLICY_GROUP Procedure&quot; on page 56-8</td>
</tr>
</tbody>
</table>
ADD_POLICY Procedure

This procedure adds a fine-grained access control policy to a table or view. 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 56-3

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); 
```

Table 56–1 DBMS_RLS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ADD_GROUPED_POLICY Procedure&quot; on page 56-9</td>
<td>Adds a policy associated with a policy group.</td>
</tr>
<tr>
<td>&quot;ADD_POLICYCONTEXT Procedure&quot; on page 56-10</td>
<td>Adds the context for the active application.</td>
</tr>
<tr>
<td>&quot;DELETE_POLICY_GROUP Procedure&quot; on page 56-11</td>
<td>Deletes a policy group.</td>
</tr>
<tr>
<td>&quot;DROP_GROUPED_POLICY Procedure&quot; on page 56-12</td>
<td>Drops a policy associated with a policy group.</td>
</tr>
<tr>
<td>&quot;DROP_POLICYCONTEXT Procedure&quot; on page 56-13</td>
<td>Drops a driving context from the object so that it will have one less driving context.</td>
</tr>
<tr>
<td>&quot;ENABLE__GROUPED_POLICY Procedure&quot; on page 56-13</td>
<td>Enables or disables a row-level group security policy.</td>
</tr>
<tr>
<td>&quot;REFRESH_GROUPED_POLICY Procedure&quot; on page 56-14</td>
<td>Reparses the SQL statements associated with a refreshed policy.</td>
</tr>
</tbody>
</table>
Parameters

Table 56–2  ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table or view (logon user, if NULL).</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view to which the policy is added.</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be added. It must be unique for the same table or view.</td>
</tr>
<tr>
<td>function_schema</td>
<td>Schema of the policy function (logon user, if NULL).</td>
</tr>
<tr>
<td>policy_function</td>
<td>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.</td>
</tr>
<tr>
<td>statement_types</td>
<td>Statement types that the policy will apply. It can be any combination of SELECT, INSERT, UPDATE, and DELETE. The default is to apply to all of these types.</td>
</tr>
<tr>
<td>update_check</td>
<td>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.</td>
</tr>
<tr>
<td>enable</td>
<td>Indicates if the policy is enabled when it is added. The default is TRUE.</td>
</tr>
</tbody>
</table>

Usage Notes

- 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:

  ```sql
  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 of view that the policy will apply.
  ```

  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 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 or view.

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 56-3

A commit is also performed at the end of the operation.

Syntax

```sql
DBMS_RLS.DROP_POLICY (  
    object_schema IN VARCHAR2 := NULL,  
    object_name IN VARCHAR2,  
    policy_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table or view (logon user if NULL).</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view.</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be dropped from the table or view.</td>
</tr>
</tbody>
</table>

56-6 Supplied PL/SQL Packages and Types Reference
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 56-3

A commit is also performed at the end of the operation.

Syntax

```sql
DBMS_RLS.REFRESH_POLICY (  
   object_schema IN VARCHAR2 := NULL,  
   object_name IN VARCHAR2 := NULL,  
   policy_name IN VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view that the policy is associated with.</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be refreshed.</td>
</tr>
</tbody>
</table>

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.
CREATE_POLICY_GROUP Procedure

See Also: Usage Notes on page 56-3

A commit is also performed at the end of the operation.

Syntax

```sql
DBMS_RLS.ENABLE_POLICY (
    object_schema IN VARCHAR2 := NULL,
    object_name IN VARCHAR2,
    policy_name IN VARCHAR2,
    enable IN BOOLEAN);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table or view (logon user if NULL).</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view that the policy is associated with.</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be enabled or disabled.</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE to enable the policy, FALSE to disable the policy.</td>
</tr>
</tbody>
</table>

CREATE_POLICY_GROUP Procedure

This procedure creates a policy group.

Syntax

```sql
DBMS_RLS.CREATE_POLICY_GROUP (
    object_schema VARCHAR2,
    object_name VARCHAR2,
    policy_group VARCHAR2 );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table or view to which the policy is added.</td>
</tr>
</tbody>
</table>
Summary of DBMS_RLS Subprograms

ADD_GROUPED_POLICY Procedure

This procedure adds a policy associated with a policy group.

Syntax

```sql
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 );
```

Parameters

**Table 56–7  ADD_GROUPED_POLICY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table or view to which the policy is added.</td>
</tr>
<tr>
<td>policy_group</td>
<td>The name of the policy group that the policy belongs to.</td>
</tr>
<tr>
<td>policy_name</td>
<td>The name of the policy; must be unique for the same table or view.</td>
</tr>
<tr>
<td>function_schema</td>
<td>The schema owning the policy function.</td>
</tr>
<tr>
<td>policy_function</td>
<td>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.</td>
</tr>
</tbody>
</table>
ADD_POLICY_CONTEXT Procedure

Note the following:

- This procedure adds a policy to the specified table or view 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 );
```

### Usage Notes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement_types</td>
<td>The list of statement types to which the policy can apply. It can be any combination of SELECT, INSERT, UPDATE, or DELETE. Optional.</td>
</tr>
<tr>
<td>update_check</td>
<td>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.</td>
</tr>
<tr>
<td>enable</td>
<td>Indicates if the policy is enable when it is added. The default is TRUE.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table or view to which the policy is added.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the driving context</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute of the driving context</td>
</tr>
</tbody>
</table>

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.emp in group access_control_group, the following command is issued:

  ```sql
  DBMS_RLS.ADD_GROUPED_POLICY('hr','emp','access_control_group','policy1','SYS', 'HR.ACCESS');
  ```

DELETE_POLICY_GROUP Procedure

This procedure deletes a policy group.
DROP_GROUPED_POLICY Procedure

Syntax

```sql
DBMS_RLS.DELETE_POLICY_GROUP (
  object_schema VARCHAR2,
  object_name VARCHAR2,
  policy_group VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table or view to which the policy is added.</td>
</tr>
<tr>
<td>policy_group</td>
<td>The name of the policy group that the policy belongs to</td>
</tr>
</tbody>
</table>

Usage Notes

Note the following:
- This procedure deletes a policy group for the specified table or view.
- No policy can be in the policy group.

DROP_GROUPED_POLICY Procedure

This procedure drops a policy associated with a policy group.

Syntax

```sql
DBMS_RLS.DROP_GROUPED_POLICY (  
  object_schema VARCHAR2,  
  object_name VARCHAR2,  
  policy_group VARCHAR2,  
  policy_name VARCHAR2 );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
</tbody>
</table>
**DROP_POLICY_CONTEXT Procedure**

This procedure drops a driving context from the object so that it will have one less driving context.

**Syntax**

```sql
DBMS_RLS.DROP_POLICY_CONTEXT (
    object_schema VARCHAR2,
    object_name VARCHAR2,
    namespace VARCHAR2,
    attribute VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the table or view to which the policy is dropped.</td>
</tr>
<tr>
<td>policy_group</td>
<td>The name of the policy group that the policy belongs to.</td>
</tr>
<tr>
<td>policy_name</td>
<td>The name of the policy.</td>
</tr>
</tbody>
</table>

**ENABLE__GROUPED_POLICY Procedure**

This procedure enables or disables a row-level group security policy.

**Syntax**

```sql
DBMS_RLS.ENABLE_GROUPED_POLICY (
    object_schema VARCHAR2,
    object_name VARCHAR2,
    group_name VARCHAR2,
    policy_name VARCHAR2,
    policy_group VARCHAR2
);
```
enable BOOLEAN);

Parameters

Table 56–12 ENABLE_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table or view with which the policy is associated.</td>
</tr>
<tr>
<td>group_name</td>
<td>The name of the group of the policy.</td>
</tr>
<tr>
<td>policy_name</td>
<td>The name of the policy to be enabled or disabled.</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE enables the policy; FALSE disables the policy.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_RLS.REFRESH_GROUPED_POLICY (  
  object_schema VARCHAR2,  
  object_name VARCHAR2,  
  group_name VARCHAR2,  
  policy_name VARCHAR2 );
```

Parameters

Table 56–13 REFRESH_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>The schema containing the table or view.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table or view with which the policy is associated.</td>
</tr>
</tbody>
</table>
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 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.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.

Example

This example illustrates the necessary steps to enforce a fine-grained access control policy.

In an Oracle HR application, PER_PEOPLE is a view for the PER_ALL_PEOPLE table, and both objects are under APPS schema.

```sql
CREATE TABLE per_all_people
   (person_id NUMBER(15),
    last_name VARCHAR2(30),
    emp_no VARCHAR2(15), ...);
CREATE VIEW per_people AS
   SELECT * FROM per_all_people;
```

There should be a security policy that limits access to the PER_PEOPLE view based on the user’s role in the company. The predicates for the policy can be generated by the secure_person function in the HR_SECURITY package. The package is under schema APPS and contains functions to support all security policies related to the HR application. Also, all the application contexts are under the APPS_SEC namespace.

```sql
CREATE PACKAGE BODY hr_security IS
   FUNCTION secure_person(obj_schema VARCHAR2, obj_name VARCHAR2) RETURN VARCHAR2 IS
      d_predicate VARCHAR2(2000);
```

---

**Table 56–13**  
**REFRESH_GROUPED_POLICY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_name</td>
<td>The name of the group of the policy.</td>
</tr>
<tr>
<td>policy_name</td>
<td>The name of the policy.</td>
</tr>
</tbody>
</table>
BEGIN
    -- for users with HR_ROLE set to EMP, map logon user name
    -- to employee id. FND_USER table stores relationship
    -- among database users, application users,
    -- and people held in the HR person table.
    IF SYS_CONTEXT('apps_sec', 'hr_role') = 'EMP' THEN
        d_predicate = 'person_id IN
            (SELECT employee_id FROM apps.fnd_user
            WHERE user_name = SYS_CONTEXT('userenv', ''session_user''))';
    ELSE IF SYS_CONTEXT('apps_sec', 'hr_role') = 'MGR' THEN
        d_predicate = 'person_id IN
            (SELECT ppl.employee_id FROM per_person_list ppl WHERE
            ppl.security_profile_id = SYS_CONTEXT('apps_sec',
            ''security_profile_id''))
            OR EXISTS (SELECT NULL FROM apps.per security_profiles psp
            WHERE
            SYS_CONTEXT('apps_sec', ''security_profile_id'') =
            psp.security_profile_id AND psp.view_all_flag = ''Y''))';
    ELSE
        d_predicate = '1=2'; -- deny access to other users, may use
        something like 'keycol=null'
    END IF;
    RETURN d_predicate;
END secure_person;
END hr_security;

The next step is to associate a policy (here we call it PER_PEOPLE_SEC) for the
PER_PEOPLE view to the HR_SECURITY.SECURE_PERSON function that generates
the dynamic predicates:

DBMS_RLS.ADD_POLICY('apps', 'per_people', 'per_people_sec', 'apps'
    'hr_security.secure_person', 'select, update, delete');

Now, any SELECT, UPDATE, and DELETE statement with the PER_PEOPLE view
involved will pick up one of the three predicates based on the value of the
application context HR_ROLE.


**Note:** The same security function that secured the `PER_ALL_ PEOPLE` table can also be used to generate the dynamic predicates to secure the `PER_ADDRESSES` table, because they have the same policy to limit access to data.
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
- Requirements
- ROWID Types
- Exceptions
- Summary of DBMS_ROWID Subprograms
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/SQL 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:

```sql
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:

```sql
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

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 Oracle8i and above.

ROWID Verification Results

VALID          Valid ROWID
INVALID        Invalid ROWID

For example:
rowid_is_valid constant integer := 0;
rowid_is_invalid constant integer := 1;

Object Types

UNDEFINED    Object Number not defined (for restricted ROWIDs)

For example:
rowid_object_undefined constant integer := 0;
ROWID Conversion Types

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

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 57–1  DBMS_ROWID Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ROWID_CREATE Function&quot; on page 57-5</td>
<td>Creates a ROWID, for testing only.</td>
</tr>
<tr>
<td>&quot;ROWID_INFO Procedure&quot; on page 57-6</td>
<td>Returns the type and components of a ROWID.</td>
</tr>
<tr>
<td>&quot;ROWID_TYPE Function&quot; on page 57-7</td>
<td>Returns the ROWID type: 0 is restricted, 1 is extended.</td>
</tr>
<tr>
<td>&quot;ROWID_OBJECT Function&quot; on page 57-8</td>
<td>Returns the object number of the extended ROWID.</td>
</tr>
<tr>
<td>&quot;ROWID_RELATIVE_FNO Function&quot; on page 57-9</td>
<td>Returns the file number of a ROWID.</td>
</tr>
</tbody>
</table>
Summary of DBMS_ROWID Subprograms

Table 57–1  DBMS_ROWID Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ROWID_BLOCK_NUMBER Function&quot; on page 57–10</td>
<td>Returns the block number of a ROWID.</td>
</tr>
<tr>
<td>&quot;ROWID_ROW_NUMBER Function&quot; on page 57–10</td>
<td>Returns the row number.</td>
</tr>
<tr>
<td>&quot;ROWID_TO_ABSOLUTE_FNO Function&quot; on page 57–11</td>
<td>Returns the absolute file number associated with the ROWID for a row in a specific table.</td>
</tr>
<tr>
<td>&quot;ROWID_TO_EXTENDED Function&quot; on page 57–12</td>
<td>Converts a ROWID from restricted format to extended.</td>
</tr>
<tr>
<td>&quot;ROWID_TO_RESTRICTED Function&quot; on page 57–14</td>
<td>Converts an extended ROWID to restricted format.</td>
</tr>
<tr>
<td>&quot;ROWID_VERIFY Function&quot; on page 57–15</td>
<td>Checks if a ROWID can be correctly extended by the ROWID_TO_EXTENDED function.</td>
</tr>
</tbody>
</table>

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

```sql
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

```sql
pragma RESTRICT_REFERENCES(rowid_create, WNDs, RNDS, WNPS, RNPS);
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_type</td>
<td>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.</td>
</tr>
<tr>
<td>object_number</td>
<td>Data object number (rowid_object_undefined for restricted).</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number.</td>
</tr>
<tr>
<td>block_number</td>
<td>Block number in this file.</td>
</tr>
<tr>
<td>file_number</td>
<td>File number in this block.</td>
</tr>
</tbody>
</table>

**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,  
)
```
Summary of DBMS_ROWID Subprograms

```sql
block_number OUT NUMBER,
row_number OUT NUMBER);
```

**Pragmas**

```sql
pragma RESTRICT_REFERENCES(rowid_info, WNDS, RNDS, WNPS, RNPS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_in</td>
<td>ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.</td>
</tr>
<tr>
<td>rowid_type</td>
<td>Returns type (restricted/extended).</td>
</tr>
<tr>
<td>object_number</td>
<td>Returns data object number (rowid_object_undefined for restricted).</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Returns relative file number.</td>
</tr>
<tr>
<td>block_number</td>
<td>Returns block number in this file.</td>
</tr>
<tr>
<td>file_number</td>
<td>Returns file number in this block.</td>
</tr>
</tbody>
</table>

**See Also:** "ROWID_TYPE Function" on page 57-7

**Example**

This example reads back the values for the ROWID that you created in the ROWID CREATE:

```sql
DBMS_ROWID.ROWID_INFO(my_rowid, rid_type, obj_num,
    file_num, block_num, row_num);
```

```sql
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.
ROWID_OBJECT Function

Syntax

```sql
DBMS_ROWID.ROWID_TYPE (
    rowid_id IN ROWID)
RETURN NUMBER;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_type, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 57–4 ROWID_TYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted.</td>
</tr>
</tbody>
</table>

Example

```sql
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

```sql
DBMS_ROWID.ROWID_OBJECT (
    rowid_id IN ROWID)
RETURN NUMBER;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_object, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 57–5 ROWID_OBJECT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted.</td>
</tr>
</tbody>
</table>
Summary of DBMS_ROWID Subprograms

**Note:** The ROWID_OBJECT_UNDEFINED constant is returned for restricted ROWIDs.

---

### Example

```plsql
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

```plsql
DBMS_ROWID.ROWID_RELATIVE_FNO (
    rowid_id IN ROWID)
RETURN NUMBER;
```

#### Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted.</td>
</tr>
</tbody>
</table>

#### Example

The example PL/SQL code fragment returns the relative file number:

```plsql
DECLARE
    file_number INTEGER;
    rowid_val ROWID;
BEGIN
    SELECT ROWID INTO rowid_val
    FROM dept
    WHERE loc = 'Boston';
    file_number :=
```
ROWID_BLOCK_NUMBER Function

This function returns the database block number for the input ROWID.

Syntax

```sql
DBMS_ROWID.ROWID_BLOCK_NUMBER (row_id IN ROWID)
RETURN NUMBER;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted.</td>
</tr>
</tbody>
</table>

Example

The example SQL statement selects the block number from a ROWID and inserts it into another table:

```sql
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

```sql
DBMS_ROWID.ROWID_ROW_NUMBER (row_id IN ROWID)
RETURN NUMBER;
```
Summary of DBMS_ROWID Subprograms

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_row_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Table 57–8 ROWID_ROW_NUMBER Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>row_id</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 57–9 ROWID_TO_ABSOLUTE_FNO Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>row_id</td>
</tr>
</tbody>
</table>
ROWID_TO_EXTENDED Function

Table 57–9  ROWID_TO_ABSOLUTE_FNO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table.</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name.</td>
</tr>
</tbody>
</table>

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 57–10  ROWID_TO_EXTENDED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_rowid</td>
<td>ROWID to be converted.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table (optional).</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name (optional).</td>
</tr>
<tr>
<td>conversion_type</td>
<td>rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type, or the character string).</td>
</tr>
</tbody>
</table>

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:

```sql
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 57–11 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.

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_in</td>
<td>ROWID to be verified.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table.</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name.</td>
</tr>
<tr>
<td>conversion_type</td>
<td>rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type or the character string).</td>
</tr>
</tbody>
</table>

**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
```
ROWID_VERIFY Function

WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;

See Also:  Chapter 80, "UTL_RAW", Chapter 81, "UTL_REF"
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET_IDENTIFIER&quot; on page 58-3</td>
<td>Sets the identifier.</td>
</tr>
<tr>
<td>&quot;SETCONTEXT&quot; on page 58-4</td>
<td>Sets the context.</td>
</tr>
<tr>
<td>&quot;CLEARCONTEXT&quot; on page 58-5</td>
<td>Clears the context.</td>
</tr>
<tr>
<td>&quot;CLEARIDENTIFIER&quot; on page 58-6</td>
<td>Clears the identifier.</td>
</tr>
<tr>
<td>&quot;SETROLE Procedure&quot; on page 58-6</td>
<td>Sets role.</td>
</tr>
<tr>
<td>&quot;SET_SQL_TRACE Procedure&quot; on page 58-7</td>
<td>Turns tracing on or off.</td>
</tr>
<tr>
<td>&quot;SET_NLS Procedure&quot; on page 58-7</td>
<td>Sets national language support (NLS).</td>
</tr>
<tr>
<td>&quot;CLOSE_DATABASE_LINK Procedure&quot; on page 58-8</td>
<td>Closes database link.</td>
</tr>
<tr>
<td>&quot;RESET_PACKAGE Procedure&quot; on page 58-8</td>
<td>Deinstantiates all packages in the session.</td>
</tr>
<tr>
<td>&quot;UNIQUE_SESSION_ID Function&quot; on page 58-10</td>
<td>Returns an identifier that is unique for all sessions currently connected to this database.</td>
</tr>
<tr>
<td>&quot;IS_ROLE_ENABLED Function&quot; on page 58-10</td>
<td>Determines if the named role is enabled for the session.</td>
</tr>
<tr>
<td>&quot;IS_SESSION_ALIVE Function&quot; on page 58-11</td>
<td>Determines if the specified session is alive.</td>
</tr>
<tr>
<td>&quot;SET_CLOSE_CACHED_OPEN_CURSORS Procedure&quot; on page 58-11</td>
<td>Turns close_cached_open_cursors on or off.</td>
</tr>
<tr>
<td>&quot;FREE_UNUSED_USER_MEMORY Procedure&quot; on page 58-12</td>
<td>Lets you reclaim unused memory after performing operations requiring large amounts of memory.</td>
</tr>
</tbody>
</table>


Table 58–1  DBMS_SESSION Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET_CONTEXT Procedure&quot; on page 58-14</td>
<td>Sets or resets the value of a context attribute.</td>
</tr>
<tr>
<td>&quot;LIST_CONTEXT Procedure&quot; on page 58-15</td>
<td>Returns a list of active namespace and context for the current session.</td>
</tr>
<tr>
<td>&quot;SWITCH_CURRENT_CONSUMER_GROUP Procedure&quot; on page 58-16</td>
<td>Facilitates changing the current resource consumer group of a user’s current session.</td>
</tr>
</tbody>
</table>

**SET_IDENTIFIER**

This procedure sets the client ID in the session.

**Syntax**

```sql
DBMS_SESSION.SET_IDENTIFIER (  
    client_id VARCHAR2);
```

**Parameters**

Table 58–2  SET_IDENTIFIER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>The application-specific identifier of the current database session.</td>
</tr>
</tbody>
</table>

**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.
SET_CONTEXT Procedure

Syntax

```sql
DBMS_SESSION.SET_CONTEXT (
    namespace VARCHAR2,
    attribute VARCHAR2,
    value   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>The namespace of the application context to be set</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute of the application context to be set</td>
</tr>
<tr>
<td>value</td>
<td>The value of the application context to be set</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SESSION.SET_CONTEXT (
    namespace VARCHAR2,
    attribute VARCHAR2,
    value   VARCHAR2,
    username VARCHAR2,
    client_id VARCHAR2 );
```
Parameters

Table 58–4  SET_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>The namespace of the application context to be set</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute of the application context to be set</td>
</tr>
<tr>
<td>value</td>
<td>The value of the application context to be set</td>
</tr>
<tr>
<td>username</td>
<td>The username attribute of the application context</td>
</tr>
<tr>
<td>client_id</td>
<td>The client_id attribute of the application context (64-byte maximum)</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SESSION.CLEAR_CONTEXT
    namespace VARCHAR2,
    attribute VARCHAR2,
    username VARCHAR2,
    client_id VARCHAR2);
```
CLEAR_IDENTIFIER

Parameters

Table 58–5  CLEAR_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>The namespace in which the application context is to be cleared. Required.</td>
</tr>
<tr>
<td>client_id</td>
<td>Applies to global context and is optional for other types of contexts; 64-byte maximum.</td>
</tr>
<tr>
<td>attribute</td>
<td>The specific attribute in the namespace to be cleared. Optional. The default is NULL; all attributes are to be considered.</td>
</tr>
<tr>
<td>username</td>
<td>The username attribute of the application context</td>
</tr>
</tbody>
</table>

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 58–6  SET_ROLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_cmd</td>
<td>This text is appended to “set role” and then run as SQL.</td>
</tr>
</tbody>
</table>

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 58–7  SET_SQL_TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_trace</td>
<td>TRUE turns tracing on, FALSE turns tracing off.</td>
</tr>
</tbody>
</table>

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);
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

```sql
DBMS_SESSION.CLOSE_DATABASE_LINK (  
   dblink VARCHAR2);
```

Parameters

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>NLS parameter. The parameter name must begin with 'NLS'.</td>
</tr>
<tr>
<td>value</td>
<td>Parameter value.</td>
</tr>
</tbody>
</table>
```

- If the parameter is a text literal, then it needs embedded single-quotes. For example, "set_nls('nls_date_format','"DD-MON-YY")"

RESET_PACKAGE Procedure

This procedure deinstantiates all packages in this session: It frees all package state.

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 recreate 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:

```
EXECUTE 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;
```
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 58–10  UNIQUE_SESSION_ID Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_session_id</td>
<td>Returns up to 24 bytes.</td>
</tr>
</tbody>
</table>

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 58–11  IS_ROLE_ENABLED Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>Name of the role.</td>
</tr>
</tbody>
</table>
Summary of DBMS_SESSION Subprograms

Returns

Table 58–12  IS_ROLE_ENABLED Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_role_enabled</td>
<td>TRUE or FALSE, depending on whether the role is enabled.</td>
</tr>
</tbody>
</table>

**IS_SESSION_ALIVE Function**

This function determines if the specified session is alive.

**Syntax**

```sql
DBMS_SESSION.IS_SESSION_ALIVE (uniqueid VARCHAR2)
RETURN BOOLEAN;
```

**Parameters**

Table 58–13  IS_SESSION_ALIVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueid</td>
<td>Unique ID of the session: This is the same one as returned by UNIQUE_SESSION_ID.</td>
</tr>
</tbody>
</table>

**Returns**

Table 58–14  IS_SESSION_ALIVE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_session_alive</td>
<td>TRUE or FALSE, depending on whether the session is alive.</td>
</tr>
</tbody>
</table>

**SET_CLOSE_CACHED_OPEN_CURSORS Procedure**

This procedure turns close_cached_open.Cursors on or off. It is equivalent to the following SQL statement:

```sql
ALTER SESSION SET CLOSE_CACHED_OPEN_CURSORS ...
```

**Syntax**

```sql
DBMS_SESSION.SET_CLOSE_CACHED_OPEN_CURSORS (close_cursors BOOLEAN);
```
FREE_UNUSED_USER_MEMORY Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close_cursors</td>
<td>TRUE or FALSE</td>
</tr>
</tbody>
</table>

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.
- **MTS 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 PL/SQL fragment below 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Name of the namespace to use for the application context (limited to 30 bytes).</td>
</tr>
<tr>
<td>attribute</td>
<td>Name of the attribute to be set (limited to 30 bytes).</td>
</tr>
<tr>
<td>value</td>
<td>Value to be set (limited to 4 kilobytes).</td>
</tr>
<tr>
<td>username</td>
<td>The username attribute of the application context</td>
</tr>
<tr>
<td>client_id</td>
<td>The application-specific identifier of the current database session.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```
SWITCH_CURRENT_CONSUMER_GROUP Procedure

Parameters

Table 58–17  LISTCONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Buffer to store a list of application context set in the current session.</td>
</tr>
</tbody>
</table>

Returns

Table 58–18  LISTCONTEXT Procedure Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list of (namespace, attribute, values) set in current session</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of entries in the buffer returned</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SESSION.switch_current_consumer_group (  
  new_consumer_group      IN    VARCHAR2,  
  old_consumer_group      OUT   VARCHAR2,  
  initial_group_on_error  IN    BOOLEAN);
```
Parameters

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 58–20 SWITCH_CURRENT_CONSUMER_GROUP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29368</td>
<td>Non-existent consumer group.</td>
</tr>
<tr>
<td>1031</td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td>29396</td>
<td>Cannot switch to OTHER_GROUPS consumer group.</td>
</tr>
</tbody>
</table>

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

```sql
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('tkrogrpl', 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;
/```
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
- Usage Notes
- Summary of DBMS_SHARED_POOL Subprograms
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>
<thead>
<tr>
<th>Table 59–1 DBMS_SHARED_POOL Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subprogram</td>
</tr>
<tr>
<td>&quot;SIZES Procedure&quot; on page 59-2</td>
</tr>
<tr>
<td>&quot;KEEP Procedure&quot; on page 59-3</td>
</tr>
<tr>
<td>&quot;UNKEEP Procedure&quot; on page 59-5</td>
</tr>
<tr>
<td>&quot;ABORTED_REQUEST_THRESHOLD Procedure&quot; on page 59-5</td>
</tr>
</tbody>
</table>

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 below.
### Syntax

```sql
DBMS_SHARED_POOL.SIZES (
    minsize NUMBER);
```

### Parameters

* **Table 59–2 SIZES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minsize</td>
<td>Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.</td>
</tr>
</tbody>
</table>

### 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

```sql
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 59–3  KEEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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.</td>
</tr>
<tr>
<td>flag</td>
<td>(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'.</td>
</tr>
</tbody>
</table>

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 (i.e., delimited identifiers, multi-byte names, etc. 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**

```sql
DBMS_SHARED_POOL.UNKEEP (name VARCHAR2,
                           flag CHAR DEFAULT 'P');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the object to unkeep. See description of the name object for the KEEP procedure.</td>
</tr>
<tr>
<td>flag</td>
<td>See description of the flag parameter for the KEEP procedure.</td>
</tr>
</tbody>
</table>

**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**

```sql
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (threshold_size NUMBER);
```
ABORTED_REQUEST_THRESHOLD Procedure

Parameters

Table 59–5  ABORTED_REQUEST_THRESHOLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold_size</td>
<td>Size, in bytes, of a request which does not try to free unpinned (not &quot;unkeep-ed&quot;) memory within the shared pool. The range of threshold_size is 5000 to ~2 GB inclusive.</td>
</tr>
</tbody>
</table>

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 threshold_size. This user gets the 'out of memory' error without attempting to search the LRU list.
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;UNUSED_SPACE Procedure&quot; on page 60-2</td>
<td>Returns information about unused space in an object (table, index, or cluster).</td>
</tr>
<tr>
<td>&quot;FREE_BLOCKS Procedure&quot; on page 60-3</td>
<td>Returns information about free blocks in an object (table, index, or cluster).</td>
</tr>
<tr>
<td>&quot;SPACE_USAGE Procedure&quot; on page 60-5</td>
<td>Returns information about free blocks in a bitmapped segment.</td>
</tr>
</tbody>
</table>

UNUSED_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed.</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the segment to be analyzed.</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed:</td>
</tr>
<tr>
<td></td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td>TABLE PARTITION</td>
</tr>
<tr>
<td></td>
<td>TABLE SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>INDEX PARTITION</td>
</tr>
<tr>
<td></td>
<td>INDEX SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>CLUSTER</td>
</tr>
<tr>
<td></td>
<td>LOB</td>
</tr>
<tr>
<td>total_blocks</td>
<td>Returns total number of blocks in the segment.</td>
</tr>
<tr>
<td>total_bytes</td>
<td>Returns total number of blocks in the segment, in bytes.</td>
</tr>
<tr>
<td>unused_blocks</td>
<td>Returns number of blocks which are not used.</td>
</tr>
<tr>
<td>unused_bytes</td>
<td>Returns, in bytes, number of blocks which are not used.</td>
</tr>
<tr>
<td>last_used_extent_file_id</td>
<td>Returns the file ID of the last extent which contains data.</td>
</tr>
<tr>
<td>last_used_extent_block_id</td>
<td>Returns the block ID of the last extent which contains data.</td>
</tr>
<tr>
<td>last_used_block</td>
<td>Returns the last block within this extent which contains data.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed.</td>
</tr>
</tbody>
</table>

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 (}
FREE_BLOCKS Procedure

```
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 60–3 FREE_BLOCKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed.</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the segment to be analyzed.</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed:</td>
</tr>
<tr>
<td>freelist_group_id</td>
<td>Freelist group (instance) whose free list size is to be computed.</td>
</tr>
<tr>
<td>free_blks</td>
<td>Returns count of free blocks for the specified group.</td>
</tr>
<tr>
<td>scan_limit</td>
<td>Maximum number of free list blocks to read (optional).</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed.</td>
</tr>
</tbody>
</table>

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 60–4  SPACE_USAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_name</td>
<td>Name of the segment to be analyzed</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER)</td>
</tr>
</tbody>
</table>

OUTPUT ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unformatted_blocks</td>
<td>Total number of blocks that are unformatted</td>
</tr>
<tr>
<td>unformatted bytes</td>
<td>Total number of bytes that are unformatted</td>
</tr>
<tr>
<td>f01_blocks</td>
<td>Number of blocks that has at least 0 to 25% free space</td>
</tr>
<tr>
<td>f01_bytes</td>
<td>Number of bytes that has at least 0 to 25% free space</td>
</tr>
<tr>
<td>f02_blocks</td>
<td>Number of blocks that has at least 25 to 50% free space</td>
</tr>
<tr>
<td>f02_bytes</td>
<td>Number of bytes that has at least 25 to 50% free space</td>
</tr>
<tr>
<td>f03_blocks</td>
<td>Number of blocks that has at least 50 to 75% free space</td>
</tr>
<tr>
<td>f03_bytes</td>
<td>Number of bytes that has at least 50 to 75% free space</td>
</tr>
<tr>
<td>f04_blocks</td>
<td>Number of blocks that has at least 75 to 100% free space</td>
</tr>
<tr>
<td>f04_bytes</td>
<td>Number of bytes that has at least 75 to 100% free space</td>
</tr>
<tr>
<td>full_blocks</td>
<td>Total number of blocks that are full in the segment</td>
</tr>
<tr>
<td>full_bytes</td>
<td>Total number of bytes that are full in the segment</td>
</tr>
</tbody>
</table>

Example

```sql
variable unf number;
variable unfb number;
variable f01 number;
variable f01b number;
variable f02 number;
variable f02b number;
variable f03 number;
variable f03b number;
variable f04 number;
variable f04b 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;
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 and Constants for DBMS_SPACE_ADMIN
- Summary of DBMS_SPACE_ADMIN Subprograms
Security and Constants for DBMS_SPACE_ADMIN

Security
This package runs with SYS privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

Constants

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGMENT_VERIFY_EXTENTS</td>
<td>Verifies that the space owned by segment is appropriately reflected in the bitmap as used.</td>
</tr>
<tr>
<td>SEGMENT_VERIFY_EXTENTS_GLOBAL</td>
<td>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.</td>
</tr>
<tr>
<td>SEGMENT_MARK_CORRUPT</td>
<td>Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation).</td>
</tr>
<tr>
<td>SEGMENT_MARK_VALID</td>
<td>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.</td>
</tr>
<tr>
<td>SEGMENT_DUMP_EXTENT_MAP</td>
<td>Dumps the extent map for a given segment.</td>
</tr>
<tr>
<td>TABLESPACE_VERIFY_BITMAP</td>
<td>Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent.</td>
</tr>
<tr>
<td>TABLESPACE_EXTENT_MAKE_FREE</td>
<td>Makes this range (extent) of space free in the bitmaps.</td>
</tr>
<tr>
<td>TABLESPACE_EXTENT_MAKE_USED</td>
<td>Makes this range (extent) of space used in the bitmaps.</td>
</tr>
</tbody>
</table>

Summary of DBMS_SPACE_ADMIN Subprograms

Table 61–1  DBMS_SPACE_ADMIN Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SEGMENT_VERIFY Procedure&quot; on page 61-3</td>
<td>Verifies the consistency of the extent map of the segment.</td>
</tr>
</tbody>
</table>
SEGMENT_VERIFY Procedure

This procedure verifies that the extent map of the segment is consistent with the bitmap.

Syntax

```sql
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (  
    tablespace_name IN VARCHAR2,  
    header_relative_file IN POSITIVE,  
    header_block IN POSITIVE,
```
### SEGMENT_CORRUPT Procedure

*verify_option* IN *POSITIVE* DEFAULT SEGMENT_VERIFY_EXTENTS);

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablespace_name</code></td>
<td>Name of tablespace in which segment resides.</td>
</tr>
<tr>
<td><code>header_relative_file</code></td>
<td>Relative file number of segment header.</td>
</tr>
<tr>
<td><code>header_block</code></td>
<td>Block number of segment header.</td>
</tr>
<tr>
<td><code>verify_option</code></td>
<td>What kind of check to do: SEGMENT_VERIFY_EXTENTS or SEGMENT_VERIFY_EXTENTS_GLOBAL.</td>
</tr>
</tbody>
</table>

#### 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.

#### Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (  
  tablespace_name       IN  VARCHAR2,  
  header_relative_file  IN  POSITIVE,  

```
Summary of DBMS_SPACE_ADMIN Subprograms

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides.</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header.</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header.</td>
</tr>
<tr>
<td>corrupt_option SEGMENT_MARK_CORRUPT (default) or SEGMENT_MARK_VALID.</td>
<td></td>
</tr>
</tbody>
</table>

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 space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS or TABLESPACE_REBUILD_BITMAPS procedure. These are described later in this chapter.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (  
    tablespace_name IN VARCHAR2,  
    header_relative_file IN POSITIVE,  
    header_block IN POSITIVE);
```
SEGMENT_DUMP Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides.</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header.</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header.</td>
</tr>
</tbody>
</table>

Example

EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides.</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header.</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header.</td>
</tr>
<tr>
<td>dump_option</td>
<td>SEGMENT_DUMP_EXTENT_MAP</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (    tablespace_name IN VARCHAR2,
    verify_option IN POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
<tr>
<td>verify_option</td>
<td>TABLESPACE_VERIFY_BITMAP</td>
</tr>
</tbody>
</table>

Example

```sql
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.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
</tbody>
</table>
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 bitmap(s). If no bitmap block DBA is specified, then it rebuilds all bitmaps for the given 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
<tr>
<td>bitmap_relative_file</td>
<td>Relative file number of bitmap block to rebuild.</td>
</tr>
<tr>
<td>bitmap_block</td>
<td>Block number of bitmap block to rebuild.</td>
</tr>
</tbody>
</table>

---

Table 61–7  TABLESPACE_FIX_BITMAPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbarange_relative_file</td>
<td>Relative file number of DBA range (extent).</td>
</tr>
<tr>
<td>dbarange_begin_block</td>
<td>Block number of beginning of extent.</td>
</tr>
<tr>
<td>dbarange_end_block</td>
<td>Block number (inclusive) of end of extent.</td>
</tr>
<tr>
<td>fix_option</td>
<td>TABLESPACE_EXTENT_MAKE_FREE or TABLESPACE_</td>
</tr>
<tr>
<td></td>
<td>EXTENT_MAKE_USED.</td>
</tr>
</tbody>
</table>

---

Table 61–8  TABLESPACE_REBUILD_BITMAPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
<tr>
<td>bitmap_relative_file</td>
<td>Relative file number of bitmap block to rebuild.</td>
</tr>
<tr>
<td>bitmap_block</td>
<td>Block number of bitmap block to rebuild.</td>
</tr>
</tbody>
</table>
Example

The following example rebuilds bitmaps for all the files in the USERS tablespace.

EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');

---

**Note:** Currently, 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
</tbody>
</table>

**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.

**Syntax**

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
    tablespace_name IN VARCHAR2);
```
TABLESPACE_MIGRATE_TO_LOCAL Procedure

Parameter

Table 61–10  TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace.</td>
</tr>
</tbody>
</table>

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.

Note: Migration of SYSTEM tablespaces will be supported in the future.

Example

EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');

TABLESPACE_MIGRATE_TO_LOCAL Procedure

Use this procedure to migrate the tablespace from dictionary managed format to locally managed format. Tablespaces migrated to locally managed format are user-managed.
Syntax

TABLESPACE_MIGRATE_TO_LOCAL(tablespace_name, allocation_unit, relative_fno)

Parameters

Table 61–11  Parameters for TABLESPACE_MIGRATE_TO_LOCAL

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Purpose</th>
<th>Datatype</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablespace Name</td>
<td>Name of the tablespace to be migrated.</td>
<td>VARCHAR</td>
<td>IN</td>
</tr>
<tr>
<td>Allocation Unit</td>
<td>Unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace.</td>
<td>INTEGER</td>
<td>IN</td>
</tr>
<tr>
<td>Relative File Number</td>
<td>Relative File Number of the file where the bitmap blocks should be placed (optional)</td>
<td>INTEGER</td>
<td>IN</td>
</tr>
</tbody>
</table>

Usage Notes

The tablespace must be kept online and read write during migration. Note that temporary tablespaces cannot be migrated. Migration of SYSTEM tablespace is not permitted in this release.

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 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 tablespace_relocate_bitmaps command to move the control information to a different destination and then resize the file.

Syntax

TABLESPACE_RELOCATE_BITMAPS (Tablespace_Name, Relative_fno, Block_Number)

Parameters

Table 61-12 Parameters for TABLESPACE_RELOCATE_BITMAPS

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Purpose</th>
<th>Datatype</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablespace_Name</td>
<td>Name of Tablespace.</td>
<td>VARCHAR</td>
<td>IN</td>
</tr>
<tr>
<td>RFNO</td>
<td>Relative File Number of the destination file.</td>
<td>NUMBER</td>
<td>IN</td>
</tr>
<tr>
<td>BLKNO</td>
<td>Block Number of the destination dba.</td>
<td>NUMBER</td>
<td>IN</td>
</tr>
</tbody>
</table>

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.
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

TABLESPACE_FIX_SEGMENT_STATES(Tablespace_Name);

Parameters

Table 61–13 Parameter for TABLESPACE_FIX_SEGMENT_STATES

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Purpose</th>
<th>Datatype</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablespace_Name</td>
<td>Name of the tablespace whose segments needs to be fixed.</td>
<td>VARCHAR</td>
<td>IN</td>
</tr>
</tbody>
</table>

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');
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 \texttt{PARSE} procedure supplied with the DBMS_SQL package.

\textbf{Note:} Oracle8i 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.

\textbf{See Also:} For more information on native dynamic SQL, see \textit{PL/SQL User's Guide and Reference}.

For a comparison of DBMS_SQL and native dynamic SQL, see \textit{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 is shown below. 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.

```sql
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);
  DBMS_SQL.EXECUTE(cursor_name);
  DBMS_SQL.FETCH_ROWS(cursor_name, rows_processed);
END;
```

See Also: Oracle Call Interface Programmer’s Guide
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.

*See Also:* Oracle8 Concepts provides an explanation of how SQL statements are parsed.

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.
Figure 62–1  DBMS_SQL Execution Flow

open_cursor

PARSE

Use bind variables? yes no

bind_variable

query? yes no

EXECUTE

PL/SQL block? yes no

variable_value

close_cursor

define_column

EXECUTE

fetch_rows

column_value

variable_value
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 per 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 8i, 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:

```sql
CREATE OR REPLACE PROCEDURE exec(STRING IN varchar2) AS
    cursor_name INTEGER;
    ret INTEGER;
BEGIN
    cursor_name := DBMS_SQL.OPEN_CURSOR;
    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:

```sql
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.

```sql
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.

```sql
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
```
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 if;
end;
/

In the example above, 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 a 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:

```plsql
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 example above 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 re-execute 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:

```sql
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:

```sql
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;
```
Examples

```sql
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 out binds 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

```sql
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

```sql
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;
```
Examples

iii) Single row delete

```plsql
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

```plsql
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.

```plsql
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 :bnd3', 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 62–1  DBMS_SQL Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;OPEN_CURSOR Function&quot; on page 62-24</td>
<td>Returns cursor ID number of new cursor.</td>
</tr>
<tr>
<td>&quot;PARSE Procedure&quot; on page 62-24</td>
<td>Parses given statement.</td>
</tr>
<tr>
<td>&quot;BIND_VARIABLE and BIND_ARRAY Procedures&quot; on page 62-27</td>
<td>Binds a given value to a given variable.</td>
</tr>
<tr>
<td>&quot;BIND_VARIABLE and BIND_ARRAY Procedures&quot; on page 62-27</td>
<td>Binds a given value to a given collection.</td>
</tr>
<tr>
<td>&quot;DEFINE_COLUMN Procedure&quot; on page 62-31</td>
<td>Defines a column to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>&quot;DEFINE_ARRAY Procedure&quot; on page 62-33</td>
<td>Defines a collection to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>&quot;DEFINE_COLUMN_LONG Procedure&quot; on page 62-35</td>
<td>Defines a LONG column to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>&quot;EXECUTE Function&quot; on page 62-35</td>
<td>Executes a given cursor.</td>
</tr>
<tr>
<td>&quot;EXECUTE_AND_FETCH Function&quot; on page 62-36</td>
<td>Executes a given cursor and fetch rows.</td>
</tr>
<tr>
<td>&quot;FETCH_ROWS Function&quot; on page 62-37</td>
<td>Fetches a row from a given cursor.</td>
</tr>
<tr>
<td>&quot;COLUMN_VALUE Procedure&quot; on page 62-37</td>
<td>Returns value of the cursor element for a given position in a cursor.</td>
</tr>
<tr>
<td>&quot;COLUMN_VALUE_LONG Procedure&quot; on page 62-40</td>
<td>Returns a selected part of a LONG column, that has been defined using DEFINE_COLUMN_LONG.</td>
</tr>
<tr>
<td>&quot;VARIABLE_VALUE Procedure&quot; on page 62-40</td>
<td>Returns value of named variable for given cursor.</td>
</tr>
<tr>
<td>&quot;IS_OPEN Function&quot; on page 62-42</td>
<td>Returns TRUE if given cursor is open.</td>
</tr>
<tr>
<td>&quot;DESCRIBE_COLUMNS Procedure&quot; on page 62-43</td>
<td>Describes the columns for a cursor opened and parsed through DBMS_SQL.</td>
</tr>
<tr>
<td>&quot;CLOSE_CURSOR Procedure&quot; on page 62-45</td>
<td>Closes given cursor and frees memory.</td>
</tr>
</tbody>
</table>
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

```plsql
DBMS_SQL.OPEN_CURSOR
    RETURN INTEGER;
```

Pragmas

```plsql
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 syntax above is 32KB.

Syntax

```sql
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.

```sql
DBMS_SQL.PARSE (  
c IN INTEGER,  
statement IN VARCHAR2S,  
lb IN INTEGER,  
ub IN INTEGER,  
lfflg IN BOOLEAN,  
language_flag IN INTEGER);
```
PARSE Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor in which to parse the statement.</td>
</tr>
<tr>
<td>statement</td>
<td>SQL statement to be parsed. Unlike PL/SQL statements, your SQL statement should not include a final semicolon. For example:</td>
</tr>
<tr>
<td></td>
<td>DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2);</td>
</tr>
<tr>
<td></td>
<td>DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab values(1)', 2);</td>
</tr>
<tr>
<td>lb</td>
<td>Lower bound for elements in the statement.</td>
</tr>
<tr>
<td>ub</td>
<td>Upper bound for elements in the statement.</td>
</tr>
<tr>
<td>lfflg</td>
<td>If TRUE, then insert a linefeed after each element on concatenation.</td>
</tr>
<tr>
<td>language_flag</td>
<td>Determines how Oracle handles the SQL statement. The following options are recognized:</td>
</tr>
<tr>
<td></td>
<td>- V6 (or 0) specifies version 6 behavior.</td>
</tr>
<tr>
<td></td>
<td>- NATIVE (or 1) specifies normal behavior for the database to which the program is connected.</td>
</tr>
<tr>
<td></td>
<td>- V7 (or 2) specifies Oracle7 behavior.</td>
</tr>
</tbody>
</table>

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
```

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.

---

**Note:**

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:
BIND_VARIABLE and BIND_ARRAY Procedures

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)

Pragma
	pragma restrict_references(bind_variable, WINDS);

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 62–3  BIND_VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind a value.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable in the statement.</td>
</tr>
<tr>
<td>value</td>
<td>Value that you want to bind to the variable in the cursor.</td>
</tr>
<tr>
<td></td>
<td>For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.</td>
</tr>
<tr>
<td>out_value_size</td>
<td>Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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 62–4  BIND_ARRAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind a value.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the collection in the statement.</td>
</tr>
<tr>
<td>table_variable</td>
<td>Local variable that has been declared as &lt;datatype&gt;.</td>
</tr>
<tr>
<td>index1</td>
<td>Index for the table element that marks the lower bound of the range.</td>
</tr>
<tr>
<td>index2</td>
<td>Index for the table element that marks the upper bound of the range.</td>
</tr>
</tbody>
</table>

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 62-12 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.
DEFINE_COLUMN Procedure

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,  

62-32  Supplied PL/SQL Packages and Types Reference
column IN RAW,
column_size IN INTEGER);

DBMS_SQL.DEFINENAME_ROWID (  
c IN INTEGER,  
position IN INTEGER,  
column IN ROWID);

Parameters

Parameters

Table 62–5  DEFINE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Value of the column being defined. The type of this value determines the type for the column being defined.</td>
</tr>
<tr>
<td>column_size</td>
<td>Maximum expected size of the column value, in bytes, for columns of type VARCHAR2, CHAR, and RAW.</td>
</tr>
</tbody>
</table>

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,  
bf_tab IN Bfile_Table,  
cnt IN INTEGER,  
lower_bound IN INTEGER);  

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind an array.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the array being defined.</td>
</tr>
<tr>
<td></td>
<td>The first column in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Type of the value being passed in for this parameter is the type of the</td>
</tr>
<tr>
<td></td>
<td>column to be defined.</td>
</tr>
<tr>
<td>column_size</td>
<td>Maximum expected size of the value in bytes for the VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>column.</td>
</tr>
</tbody>
</table>

The count has to 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.
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

```sql
DBMS_SQL.DEFINE_COLUMN_LONG (    c IN INTEGER,    position IN INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column in a statement has position 1.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SQL.EXECUTE (    c IN INTEGER)    RETURN INTEGER;
```
EXECUTE_AND_FETCH Function

Parameters

Table 62–8  EXECUTE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor ID number of the cursor to execute.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_SQL.EXECUTE_AND_FETCH (
    c IN INTEGER,
    exact IN BOOLEAN DEFAULT FALSE)
RETURN INTEGER;
```

Pragmas

```sql
pragma restrict_references(execute_and_fetch, WDNS);
```

Parameters

Table 62–9  EXECUTE_AND_FETCH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to execute and fetch.</td>
</tr>
<tr>
<td>exact</td>
<td>Set to TRUE to raise an exception if the number of rows actually matching the query differs from one.</td>
</tr>
<tr>
<td></td>
<td>Note: Oracle does not support the exact fetch TRUE option with LONG columns.</td>
</tr>
<tr>
<td></td>
<td>Even if an exception is raised, the rows are still fetched and available.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_SQL.FETCH_ROWS (
    c IN INTEGER)
RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number.</td>
</tr>
</tbody>
</table>

Pragmas

```sql
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

```sql
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
COLUMN_VALUE Procedure

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which you are fetching the values.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the cursor.</td>
</tr>
<tr>
<td></td>
<td>The first column in a statement has position 1.</td>
</tr>
<tr>
<td>value</td>
<td>Returns the value at the specified column and row.</td>
</tr>
<tr>
<td></td>
<td>If the row number specified is greater than the total number of rows fetched, then you receive an error message.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>table_variable</td>
<td>Local variable that has been declared &lt;datatype&gt;.</td>
</tr>
<tr>
<td>column_error</td>
<td>Returns any error code for the specified column value.</td>
</tr>
<tr>
<td>actual_length</td>
<td>The actual length, before any truncation, of the value in the specified column.</td>
</tr>
</tbody>
</table>

**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

```sql
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

```sql
pragma restrict_references(column_value_long,RNDS,WNDS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor ID number of the cursor from which to get the value.</td>
</tr>
<tr>
<td>position</td>
<td>Position of the column of which to get the value.</td>
</tr>
<tr>
<td>length</td>
<td>Number of bytes of the long value to fetch.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset into the long field for start of fetch.</td>
</tr>
<tr>
<td>value</td>
<td>Value of the column as a VARCHAR2.</td>
</tr>
<tr>
<td>value_length</td>
<td>Number of bytes actually returned in value.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```
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 62–14 IS_OPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor ID number of the cursor to check.</td>
</tr>
</tbody>
</table>
```
Returns

Table 62–15  IS_OPEN Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Given cursor is currently open.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Given cursor is currently not open.</td>
</tr>
</tbody>
</table>

DESCRIPT_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:

```plaintext
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 62–16  DESC_REC Type Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_type</td>
<td>Type of the column being described.</td>
</tr>
<tr>
<td>col_max_len</td>
<td>Maximum length of the column.</td>
</tr>
<tr>
<td>col_name</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>col_name_len</td>
<td>Length of the column name.</td>
</tr>
<tr>
<td>col_schema_name</td>
<td>Name of the schema the column type was defined in, if an object type.</td>
</tr>
<tr>
<td>col_schema_name_len</td>
<td>Length of the schema.</td>
</tr>
<tr>
<td>col_precision</td>
<td>Column precision, if a number.</td>
</tr>
<tr>
<td>col_scale</td>
<td>Column scale, if a number.</td>
</tr>
<tr>
<td>col_charsetid</td>
<td>Column character set identifier.</td>
</tr>
<tr>
<td>col_charsetform</td>
<td>Column character set form.</td>
</tr>
<tr>
<td>col_null_ok</td>
<td>True if column can be null.</td>
</tr>
</tbody>
</table>

The DESC_TAB Type

The DESC_TAB type is a PL/SQL table of DESC_REC records:

```plsql
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

```plsql
DBMS_SQL.DESCRIBE_COLUMNS (  
c  IN  INTEGER,  
col_cnt  OUT  INTEGER,  
desc_t  OUT  DESC_TAB);
```
Parameters

**Table 62–17  DBMS_SQL.DESCRIBE_COLUMNS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the columns being described.</td>
</tr>
<tr>
<td>col_cnt</td>
<td>Number of columns in the select list of the query.</td>
</tr>
<tr>
<td>desc_t</td>
<td>Table of DESC_REC, each DESC_REC describing a column in the query.</td>
</tr>
</tbody>
</table>

**See Also:** "Example 8: Describe Columns" on page 62-17 illustrates how to use DESCRIBE_COLUMNS.

**CLOSE_CURSOR Procedure**

This procedure closes a given cursor.

**Syntax**

```sql
DBMS_SQL.CLOSE_CURSOR (  
   c IN OUT INTEGER);
```

**Pragmas**

```sql
pragma restrict_references(close_cursor,RNDS,WNDS);
```

**Parameters**

**Table 62–18  CLOSE_CURSOR Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>IN</td>
<td>ID number of the cursor that you want to close.</td>
</tr>
<tr>
<td>c</td>
<td>OUT</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.
LAST_ROW_COUNT Function

Syntax

```sql
DBMS_SQL.LAST_ERROR_POSITION
RETURN INTEGER;
```

Pragmas

```sql
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

```sql
DBMS_SQL.LAST_ROW_COUNT
RETURN INTEGER;
```

Pragmas

```sql
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

```sql
DBMS_SQL.LAST_ROW_ID
RETURN ROWID;
```

Pragmas

```sql
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, WDNS);
```

Usage Notes

You should call this function immediately after the SQL statement is run; otherwise, the return value is undefined.
DBMS_STATS provides a mechanism for you to 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. 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
- Setting or Getting Statistics
- Gathering Optimizer Statistics
- Transferring Statistics
- Summary of DBMS_STATS Subprograms
Using DBMS_STATS

The DBMS_STATS subprograms perform the following general functions:

- PREPARE_COLUMN_VALUES Procedure
- Transferring Statistics
- Gathering 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). Users may create multiple tables with different stattab identifiers to hold separate sets of statistics.

Additionally, users can maintain different sets of statistics within a single stattab by using the statid parameter, which can help avoid cluttering the user's schema.

For all of the SET or GET procedures, if stattab is not provided (i.e., NULL), then the operation works directly on the dictionary statistics; therefore, users 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.

Most of the procedures in this package commit the current transaction, perform the operation, and then commit again. These include:

- SET_*
- DELETE_*
- EXPORT_*
- IMPORT_*
- GATHER_*
- *_STAT_TABLE

Types
Types for minimum/maximum values and histogram endpoints:

```
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:  

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;  

The constant used to indicate that auto-sample size algorithms should be used is:  
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

The following procedures enable the storage and retrieval of individual column-, index-, and table-related statistics:

PREPARE_COLUMN_VALUES  
SET_COLUMN_STATS  
SET_INDEX_STATS  
SET_SYSTEM_STATS  
SET_TABLE_STATS  

CONVERT_RAW_VALUE  
GET_COLUMN_STATS  
GET_INDEX_STATS  
GET_SYSTEM_STATS
Gathering Optimizer Statistics

The following procedures enable the gathering of 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 `statown`, `stattab`, and `statid` parameters instruct the package to backup current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating some statistics for derived objects when we have sufficient statistics on related objects:

- `GENERATE_STATS`

Transferring Statistics

The following procedures enable the transference of 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`
- `EXPORT_COLUMN_STATS`
- `EXPORT_INDEX_STATS`
- `EXPORT_SYSTEM_STATS`
- `EXPORT_TABLE_STATS`
- `EXPORT_SCHEMA_STATS`
- `EXPORT_DATABASE_STATS`
- `IMPORT_COLUMN_STATS`
- `IMPORT_INDEX_STATS`
Summary of DBMS_STATS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREPARE_COLUMN_VALUES Procedure on page 63-7</td>
<td>Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle’s internal representation for future storage via SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>SET_COLUMN_STATS Procedure on page 63-10</td>
<td>Sets column-related information.</td>
</tr>
<tr>
<td>SET_INDEX_STATS Procedure on page 63-11</td>
<td>Sets index-related information.</td>
</tr>
<tr>
<td>SET_SYSTEM_STATS Procedure on page 63-13</td>
<td>Sets system statistics.</td>
</tr>
<tr>
<td>SET_TABLE_STATS Procedure on page 63-14</td>
<td>Sets table-related information.</td>
</tr>
<tr>
<td>CONVERT_RAW_VALUE Procedure on page 63-15</td>
<td>Convert the internal representation of a minimum or maximum value into a datatype-specific value.</td>
</tr>
<tr>
<td>GET_COLUMN_STATS Procedure on page 63-16</td>
<td>Gets all column-related information.</td>
</tr>
<tr>
<td>GET_INDEX_STATS Procedure on page 63-17</td>
<td>Gets all index-related information.</td>
</tr>
<tr>
<td>GET_SYSTEM_STATS Procedure on page 63-19</td>
<td>Gets system statistics from stattab, or from the dictionary if stattab is null.</td>
</tr>
<tr>
<td>GET_TABLE_STATS Procedure on page 63-20</td>
<td>Gets all table-related information.</td>
</tr>
<tr>
<td>DELETE_COLUMN_STATS Procedure on page 63-21</td>
<td>Deletes column-related statistics.</td>
</tr>
<tr>
<td>DELETE_INDEX_STATS Procedure on page 63-22</td>
<td>Deletes index-related statistics.</td>
</tr>
<tr>
<td>DELETE_SYSTEM_STATS Procedure on page 63-23</td>
<td>Deletes system statistics.</td>
</tr>
</tbody>
</table>
### DBMS_STATS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DELETE_TABLE_STATSProcedure&quot; on page 63-24</td>
<td>Deletes table-related statistics.</td>
</tr>
<tr>
<td>&quot;DELETE_SCHEMA_STATSProcedure&quot; on page 63-25</td>
<td>Deletes schema-related statistics.</td>
</tr>
<tr>
<td>&quot;DELETE_DATABASE_STATSProcedure&quot; on page 63-26</td>
<td>Deletes statistics for the entire database.</td>
</tr>
<tr>
<td>&quot;CREATE_STAT_TABLEProcedure&quot; on page 63-27</td>
<td>Creates a table with name $stattab$ in ownname’s schema which is capable of holding statistics.</td>
</tr>
<tr>
<td>&quot;DROP_STAT_TABLE Procedure&quot; on page 63-28</td>
<td>Drops a user stat table created by CREATE_STAT_TABLE.</td>
</tr>
<tr>
<td>&quot;EXPORT_COLUMN_STATSProcedure&quot; on page 63-28</td>
<td>Retrieves statistics for a particular column and stores them in the user stat table identified by $stattab$.</td>
</tr>
<tr>
<td>&quot;EXPORT_INDEX_STATSProcedure&quot; on page 63-29</td>
<td>Retrieves statistics for a particular index and stores them in the user stat table identified by $stattab$.</td>
</tr>
<tr>
<td>&quot;EXPORT_SYSTEM_STATSProcedure&quot; on page 63-30</td>
<td>Retrieves system statistics and stores them in the user stat table.</td>
</tr>
<tr>
<td>&quot;EXPORT_TABLE_STATSProcedure&quot; on page 63-31</td>
<td>Retrieves statistics for a particular table and stores them in the user stat table.</td>
</tr>
<tr>
<td>&quot;EXPORT_SCHEMA_STATSProcedure&quot; on page 63-32</td>
<td>Retrieves statistics for all objects in the schema identified by ownname and stores them in the user stat table identified by $stattab$.</td>
</tr>
<tr>
<td>&quot;EXPORT_DATABASE_STATSProcedure&quot; on page 63-33</td>
<td>Retrieves statistics for all objects in the database and stores them in the user stat table identified by $statown$.$stattab$.</td>
</tr>
<tr>
<td>&quot;IMPORT_COLUMN_STATSProcedure&quot; on page 63-33</td>
<td>Retrieves statistics for a particular column from the user stat table identified by $stattab$ and stores them in the dictionary.</td>
</tr>
<tr>
<td>&quot;IMPORT_INDEX_STATSProcedure&quot; on page 63-34</td>
<td>Retrieves statistics for a particular index from the user stat table identified by $stattab$ and stores them in the dictionary.</td>
</tr>
<tr>
<td>&quot;IMPORT_SYSTEM_STATSProcedure&quot; on page 63-35</td>
<td>Retrieves system statistics from the user stat table and stores them in the dictionary.</td>
</tr>
<tr>
<td>&quot;IMPORT_TABLE_STATSProcedure&quot; on page 63-36</td>
<td>Retrieves statistics for a particular table from the user stat table identified by $stattab$ and stores them in the dictionary.</td>
</tr>
</tbody>
</table>
### Table 63–1 DBMS_STATS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IMPORT_SCHEMA_STATS Procedure&quot; on page 63-37</td>
<td>Retrieves statistics for all objects in the schema identified by ownername from the user stat table and stores them in the dictionary.</td>
</tr>
<tr>
<td>&quot;IMPORT_DATABASE_STATS Procedure&quot; on page 63-38</td>
<td>Retrieves statistics for all objects in the database from the user stat table and stores them in the dictionary.</td>
</tr>
<tr>
<td>&quot;GATHER_TABLE_STATS Procedure&quot; on page 63-40</td>
<td>Gathers table and column (and index) statistics.</td>
</tr>
<tr>
<td>&quot;GATHER_SCHEMA_STATS Procedure&quot; on page 63-42</td>
<td>Gathers statistics for all objects in a schema.</td>
</tr>
<tr>
<td>&quot;GATHER_DATABASE_STATS Procedure&quot; on page 63-44</td>
<td>Gathers statistics for all objects in the database.</td>
</tr>
<tr>
<td>&quot;GATHER_SYSTEM_STATS Procedure&quot; on page 63-47</td>
<td>Gathers system statistics.</td>
</tr>
<tr>
<td>&quot;GENERATE_STATS Procedure&quot; on page 63-47</td>
<td>Generates object statistics from previously collected statistics of related objects.</td>
</tr>
<tr>
<td>&quot;FLUSH_SCHEMA_MONITORING_INFO Procedure&quot; on page 63-51</td>
<td>Flushes in-memory monitoring information for the tables in the specified schema in the dictionary.</td>
</tr>
<tr>
<td>&quot;FLUSH_DATABASE_MONITORING_INFO Procedure&quot; on page 63-51</td>
<td>Flushes in-memory monitoring information for all the tables to the dictionary.</td>
</tr>
<tr>
<td>&quot;ALTER_SCHEMA_TABLE_MONITORING Procedure&quot; on page 63-52</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;ALTER_DATABASE_TABLE_MONITORING Procedure&quot; on page 63-52</td>
<td>Enables or disables the S’DML monitoring feature of all the tables in the database, except for snapshot logs and the tables, which monitoring does not support.</td>
</tr>
</tbody>
</table>

#### 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 via SET_COLUMN_STATS.
PREPARE_COLUMN_VALUES Procedure

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,  
umvals   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 63–2  PREPARE_COLUMN_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.epc</td>
<td>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.</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>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.</td>
</tr>
<tr>
<td>charvals</td>
<td>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.</td>
</tr>
<tr>
<td>datevals</td>
<td>The array of values when the column type is date-based.</td>
</tr>
<tr>
<td>numvals</td>
<td>The array of values when the column type is numeric-based.</td>
</tr>
<tr>
<td>rawvals</td>
<td>The array of values when the column type is RAW. Up to the first 32 bytes of each strings should be provided.</td>
</tr>
<tr>
<td>nvmin, nvmax</td>
<td>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.</td>
</tr>
</tbody>
</table>
SET_COLUMN_STATS Procedure

\[ \text{rwmin, rwmax} \] The minimum and maximum values when the column type is rowid. No histogram information can be provided for a column of this type.

Output parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.minval</td>
<td>Internal representation of the minimum which is suitable for use in a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.maxval</td>
<td>Internal representation of the maximum which is suitable for use in a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.novals</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20001: Invalid or inconsistent input values.

SET_COLUMN_STATS Procedure

This procedure sets column-related information.

Syntax

```sql
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);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>distcnt</td>
<td>Number of distinct values.</td>
</tr>
<tr>
<td>density</td>
<td>Column density. If this value is NULL and if distcnt is not NULL, then density is derived from distcnt.</td>
</tr>
<tr>
<td>nullcnt</td>
<td>Number of NULLs.</td>
</tr>
<tr>
<td>srec</td>
<td>StatRec structure filled in by a call to PREPARE_COLUMN_VALUES or GET_COLUMN_STATS.</td>
</tr>
<tr>
<td>avgclen</td>
<td>Average length for the column (in bytes).</td>
</tr>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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.

Syntax

```sql
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,
umlblks 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);

Parameters

Table 63–5  SET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the index (partition).</td>
</tr>
<tr>
<td>numlblks</td>
<td>Number of leaf blocks in the index (partition).</td>
</tr>
<tr>
<td>numdist</td>
<td>Number of distinct keys in the index (partition).</td>
</tr>
<tr>
<td>avglblk</td>
<td>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.</td>
</tr>
<tr>
<td>avgdblk</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 63–5 SET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clstfct</td>
<td>See clustering_factor column of the user_indexes view for a description.</td>
</tr>
<tr>
<td>indlevel</td>
<td>Height of the index (partition).</td>
</tr>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

Table 63–6 SET_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Parameter name to get, which, which can have one of the following values: sreadtim (wait time to read single block, in milliseconds); mreadtim (wait time to read a multiblock, in milliseconds); cpuspeed (cycles per second, in millions).</td>
</tr>
<tr>
<td>pvalue</td>
<td>Parameter value to get</td>
</tr>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table where the statistics will be obtained. If stattab is null, the statistics will be obtained from the dictionary.</td>
</tr>
</tbody>
</table>
SET_TABLE_STATS Procedure

**Table 63–6  SET_SYSTEM_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>The schema containing stattab, if different from the user’s schema.</td>
</tr>
</tbody>
</table>

**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**

```sql
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);  
```

**Parameters**

**Table 63–7  SET_TABLE_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

Table 63-7  SET_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the table (partition).</td>
</tr>
<tr>
<td>numblks</td>
<td>Number of blocks the table (partition) occupies.</td>
</tr>
<tr>
<td>avgrlen</td>
<td>Average row length for the table (partition).</td>
</tr>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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);
```
GET_COLUMN_STATS Procedure

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>
<thead>
<tr>
<th>Table 63–8 CONVERT_RAW_VALUE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>rawval</td>
</tr>
<tr>
<td>resval</td>
</tr>
</tbody>
</table>

GET_COLUMN_STATS Procedure

This procedure gets all column-related information.

Syntax

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);

Parameters

Table 63–9  GET_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>distcnt</td>
<td>Number of distinct values.</td>
</tr>
<tr>
<td>density</td>
<td>Column density.</td>
</tr>
<tr>
<td>nullcnt</td>
<td>Number of NULLs.</td>
</tr>
<tr>
<td>srec</td>
<td>Structure holding internal representation of column minimum, maximum, and histogram values.</td>
</tr>
<tr>
<td>avgclen</td>
<td>Average length of the column (in bytes).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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.
GET_INDEX_STATS Procedure

Syntax

```sql
DBMS_STATS.GET_INDEX_STATS (
  ownname  VARCHAR2,
  indname   VARCHAR2,
  partname  VARCHAR2 DEFAULT NULL,
  stattab   VARCHAR2 DEFAULT NULL,
  statid    VARCHAR2 DEFAULT NULL,
  numrows   OUT NUMBER,
  numblks   OUT NUMBER,
  numdist   OUT NUMBER,
  avglblk   OUT NUMBER,
  avgdblk   OUT NUMBER,
  clstfct   OUT NUMBER,
  indlevel  OUT NUMBER,
  statown   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 63–10  GET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the index (partition).</td>
</tr>
<tr>
<td>numblks</td>
<td>Number of leaf blocks in the index (partition).</td>
</tr>
<tr>
<td>numdist</td>
<td>Number of distinct keys in the index (partition).</td>
</tr>
<tr>
<td>avglblk</td>
<td>Average integral number of leaf blocks in which each distinct key appears for this index (partition).</td>
</tr>
<tr>
<td>avgdblk</td>
<td>Average integral number of data blocks in the table pointed to by a distinct key for this index (partition).</td>
</tr>
<tr>
<td>clstfct</td>
<td>Clustering factor for the index (partition).</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.  
```
GET_TABLE_STATS Procedure

This procedure gets all table-related information.

Syntax

```sql
DBMS_STATS.GET_TABLE_STATS ( 
  ownname VARCHAR2,
  tabname VARCHAR2,
  partname VARCHAR2 DEFAULT NULL,
  stattab VARCHAR2 DEFAULT NULL,
);```

Table 63–11 GET_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstop (OUT)</td>
<td>Date when statistics gathering stopped. If status = COMPLETE, the finish</td>
</tr>
<tr>
<td></td>
<td>date is returned. If status = AUTOGATHERING, the future finish date is</td>
</tr>
<tr>
<td></td>
<td>returned. If status = BADSTATS, the had-to-be-finished-by date is returned.</td>
</tr>
<tr>
<td>pname</td>
<td>The parameter name to get, which can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>sreadtim (wait time to read single block, in milliseconds); mreadtim (wait</td>
</tr>
<tr>
<td></td>
<td>time to read a multiblock, in milliseconds); cpuspeed (cycles per second,</td>
</tr>
<tr>
<td></td>
<td>in millions).</td>
</tr>
<tr>
<td>pvalue</td>
<td>The parameter value to get</td>
</tr>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table where the statistics will be obtained. If</td>
</tr>
<tr>
<td></td>
<td>stattab is null, the statistics will be obtained from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>The schema containing stattab, if different from the user’s schema.</td>
</tr>
</tbody>
</table>

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.
Summary of DBMS_STATS Subprograms

Parameters

Table 63–12  GET_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the table (partition).</td>
</tr>
<tr>
<td>numblks</td>
<td>Number of blocks the table (partition) occupies.</td>
</tr>
<tr>
<td>avgrlen</td>
<td>Average row length for the table (partition).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_STATS.DELETE_COLUMN_STATS (  
    ownname VARCHAR2,  
    tabname VARCHAR2,  
    partname VARCHAR2,  
    stattab VARCHAR2 DEFAULT NULL,  
    numrows OUT NUMBER,  
    numblks OUT NUMBER,  
    avgrlen OUT NUMBER,  
    statown VARCHAR2 DEFAULT NULL);
```
DELETE_INDEX_STATS Procedure

```sql
colname VARCHAR2,
partname VARCHAR2 DEFAULT NULL,
stattab VARCHAR2 DEFAULT NULL,
statid VARCHAR2 DEFAULT NULL,
cascade_parts BOOLEAN DEFAULT TRUE,
statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>cascade_parts</td>
<td>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.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_INDEX_STATS Procedure

This procedure deletes index-related statistics.

Syntax

```sql
DBMS_STATS.DELETE_INDEX_STATS ( 
    ownname     VARCHAR2, 
    indname     VARCHAR2,
```
Summary of DBMS_STATS Subprograms

partname VARCHAR2 DEFAULT NULL,
stattab VARCHAR2 DEFAULT NULL,
statid VARCHAR2 DEFAULT NULL,
cascade_parts BOOLEAN DEFAULT TRUE,
statown VARCHAR2 DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Table 63–14</th>
<th>DELETE_INDEX_STATS Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index.</td>
</tr>
<tr>
<td>partname</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>cascade_parts</td>
<td>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.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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);  

DELETE_TABLE_STATS Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table where the statistics will be saved.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>The schema containing stattab, if different from the user’s schema.</td>
</tr>
</tbody>
</table>

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

```sql
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);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
</tbody>
</table>
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);
```
DELETE_DATABASE_STATS Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_DATABASE_STATS Procedure

This procedure deletes statistics for an entire database.

Syntax

```sql
DBMS_STATS.DELETE_DATABASE_STATS (
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

### 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

```sql
DBMS_STATS.CREATE_STAT_TABLE (
    ownname VARCHAR2,
    stattab VARCHAR2,
    tblspace VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ownname</code></td>
<td>Name of the schema.</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>Name of the table to create. This value should be passed as the <code>stattab</code> parameter to other procedures when the user does not want to modify the dictionary statistics directly.</td>
</tr>
<tr>
<td><code>tblspace</code></td>
<td>Tablespace in which to create the stat tables. If none is specified, then they are created in the user’s default tablespace.</td>
</tr>
</tbody>
</table>

#### 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

```sql
DBMS_STATS.DROP_STAT_TABLE (
    ownname VARCHAR2,
    stattab VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```
Export Index Stats Procedure

This procedure retrieves statistics for a particular index and stores them in the user stat table identified by `stattab`.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if <code>partname</code> is NULL, then global and partition column statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within <code>stattab</code>.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different than <code>ownname</code>).</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges.
EXPORT_SYSTEM_STATS Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_STATS.EXPORT_SYSTEM_STATS (  
    stattab VARCHAR2,  
    statid VARCHAR2 DEFAULT NULL,  
    statown VARCHAR2 DEFAULT NULL);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table that describes where the statistics will be stored.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics stored from the stattab.</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

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

```sql
DBMS_STATS.EXPORT_TABLE_STATS (ownername VARCHAR2,
tabname VARCHAR2,
partname VARCHAR2 DEFAULT NULL,
stattab VARCHAR2,
statid VARCHAR2 DEFAULT NULL,
cascade BOOLEAN DEFAULT TRUE,
statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 63–24 EXPORT_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics.</td>
</tr>
</tbody>
</table>
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

```sql
DBMS_STATS.EXPORT_SCHEMA_STATS (
    ownname VARCHAR2,
    stattab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_STATS.EXPORT_SCHEMA_STATS (
    ownname VARCHAR2,
    stattab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to store the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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.statstab.

Syntax

```sql
DBMS_STATS.EXPORT_DATABASE_STATS (
    statstab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statstab</td>
<td>User stat table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within statstab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing statstab. If statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name statstab.</td>
</tr>
</tbody>
</table>

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 statstab and stores them in the dictionary.

Syntax

```sql
DBMS_STATS.IMPORT_COLUMN_STATS (
    ownname VARCHAR2,
    tabname VARCHAR2,
    colname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    statstab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```
IMPORT_INDEX_STATS Procedure

Parameters

Table 63–27 IMPORT_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition column statistics are imported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_STATS.IMPORT_INDEX_STATS(
    ownname VARCHAR2,
    indname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    stattab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```
Parameters

**Table 63–28 IMPORT_INDEX_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are imported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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**

```sql
DBMS_STATS.IMPORT_SYSTEM_STATS (  
    stattab VARCHAR2,  
    statid VARCHAR2 DEFAULT NULL,  
    statown VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 63–29 IMPORT_SYSTEM_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table where the statistics will be retrieved.</td>
</tr>
</tbody>
</table>
IMPORT_TABLE_STATS Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics retrieved from the stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>The schema containing stattab, if different from the user’s schema.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```

Parameters

Table 63–30  IMPORT_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table.</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

**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**

```sql
DBMS_STATS.IMPORT_SCHEMA_STATS (  
    ownname VARCHAR2,  
    stattab VARCHAR2,  
    statid VARCHAR2 DEFAULT NULL,  
    statown VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ownname</code></td>
<td>Name of the schema.</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
</tbody>
</table>

---

**Table 63–30 IMPORT_TABLE_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>partname</code></td>
<td>Name of the table partition. If the table is partitioned and if <code>partname</code> is NULL, then global and partition table statistics are imported.</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
<tr>
<td><code>statid</code></td>
<td>Identifier (optional) to associate with these statistics within <code>stattab</code>.</td>
</tr>
<tr>
<td><code>cascade</code></td>
<td>If true, then column and index statistics for this table are also imported.</td>
</tr>
<tr>
<td><code>statown</code></td>
<td>Schema containing <code>stattab</code> (if different than <code>ownname</code>).</td>
</tr>
</tbody>
</table>

**Table 63–31 IMPORT_SCHEMA_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ownname</code></td>
<td>Name of the schema.</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
</tbody>
</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

```sql
DBMS_STATS.IMPORT_DATABASE_STATS (
  stattab VARCHAR2,
  statid VARCHAR2 DEFAULT NULL,
  statown VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User stat table identifier describing from where to retrieve the statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

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 does not execute in parallel.

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);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of index to analyze.</td>
</tr>
<tr>
<td>indname</td>
<td>Name of index.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of partition.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to save the current statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Index does not exist or insufficient privileges.
ORA-20001: Bad input value.
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

```sql
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);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of table to analyze.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of table.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of partition.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>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.</td>
</tr>
<tr>
<td>block_sample</td>
<td>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.</td>
</tr>
</tbody>
</table>
Exceptions

ORA-20000: Table does not exist or insufficient privileges.
ORA-20001: Bad input value.

Table 63–34  GATHER_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_opt</td>
<td>Accepts: FOR ALL [INDEXED</td>
</tr>
<tr>
<td>degree</td>
<td>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.</td>
</tr>
<tr>
<td>cascade</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to save the current statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>
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);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema to analyze (NULL means current schema).</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>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.</td>
</tr>
<tr>
<td>block_sample</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 63–35  GATHER_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_opt</td>
<td>Accepts: FOR ALL [INDEXED</td>
</tr>
<tr>
<td>degree</td>
<td>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.</td>
</tr>
<tr>
<td>cascade</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>User stat table identifier describing where to save the current statistics.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
</tbody>
</table>
GATHER_DATABASE_STATS Procedure

**Syntax**

```
DBMS_STATS.GATHER_DATABASE_STATS (  
estimate_percent NUMBER DEFAULT NULL,  
block_sample BOOLEAN DEFAULT FALSE,  
```

**Table 63–35  GATHER_SCHEMA_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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). |

**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);

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);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate_percent</td>
<td>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.</td>
</tr>
<tr>
<td>block_sample</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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),  
where integer is in the range [1,254]. This value is passed to all of the individual tables. |
## Table 63–36  GATHER_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>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.</td>
</tr>
<tr>
<td>cascade</td>
<td>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.</td>
</tr>
<tr>
<td>stattab</td>
<td>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.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab.</td>
</tr>
</tbody>
</table>
Summary of DBMS_STATS Subprograms

**Exceptions**

ORA-20000: Insufficient privileges.

ORA-20001: Bad input value.

**GATHER_SYSTEM_STATS Procedure**

This procedure gathers system statistics.

**Syntax**

```sql
DBMS_STATS.GATHER_SYSTEM_STATS (
    gathering_mode VARCHAR2 DEFAULT 'INTERVAL',
    interval INTEGER DEFAULT 60,
)
```

---

**Table 63–36  GATHER_DATABASE_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>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.</td>
</tr>
<tr>
<td>objlist</td>
<td>List of objects found to be stale or empty.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>
GENERATE_STATS Procedure

```sql
STATTAB VARCHAR2 DEFAULT NULL,
STATID VARCHAR2 DEFAULT NULL,
STATOWN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gathering_mode</td>
<td>Mode values are:&lt;br&gt;INTERVAL: Captures system activity during a specified interval. The user can provide an interval value in minutes, after which system statistics will be created or updated in the dictionary or stattab.&lt;br&gt;START</td>
</tr>
<tr>
<td>interval</td>
<td>Time, in minutes, to gather statistics.</td>
</tr>
<tr>
<td>stattab</td>
<td>Identifier of the user stat table where the statistics will be saved.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab.</td>
</tr>
<tr>
<td>statown</td>
<td>The schema containing stattab, if different from the user’s schema.</td>
</tr>
</tbody>
</table>

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

```sql
DBMS_STATS.GENERATE_STATS (  
    ownname VARCHAR2,
    objname VARCHAR2,
    organized NUMBER DEFAULT 7);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of object.</td>
</tr>
<tr>
<td>objname</td>
<td>Name of object.</td>
</tr>
<tr>
<td>organized</td>
<td>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.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Unsupported object type of object does not exist.
ORA-20001: Invalid option or invalid statistics.

Example 1

Assume many modifications have been made to the `emp` 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:

```sql
BEGIN  
    DBMS_STATS.CREATE_STAT_TABLE ('scott', 'savestats');  
    DBMS_STATS.GATHER_TABLE_STATS ('scott', 'emp', 5, stattab => 'savestats');  
END;
```
This operation gathers new statistics on `emp`, but first saves the original statistics in a user stat table: `emp.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 ('scott', 'emp');
  DBMS_STATS.IMPORT_TABLE_STATS ('scott', 'emp', stattab => 'savestats');
END;
```

**Example 2**

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 stats 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 stats 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 + l');
  COMMIT;
END;
```
BEGIN
    DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS ('mystats',''OLAP'');'
    sysdate + 0.5, 'sysdate + 1');
    COMMIT;
END;

**FLUSH_SCHEMA_MONITORING_INFO Procedure**

This procedure flushes in-memory monitoring information for the tables in the specified schema to the dictionary.

**Syntax**

```sql
DBMS_STATS.FLUSH_SCHEMA_MONITORING_INFO (
    ownname varchar2 default null);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>The name of the schema. (NULL means the current schema.)</td>
</tr>
</tbody>
</table>

**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.

**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. Equivalent to issuing ALTER TABLE ... MONITORING (or NOMONITORING) individually. It is recommended to enable monitoring if you use gather_database_stats or gather_schema_stats with GATHER AUTO or GATHER STALE option.

**Syntax**

```sql
DBMS_STATS.alter_schema_table_monitoring (    
    ownname varchar2 default NULL, 
    monitoring boolean default TRUE);
```

**Parameters**

**Table 63–40  ALTER_SCHEMA_TABLE_MONITORING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>The name of the schema. (NULL means the current schema.)</td>
</tr>
<tr>
<td>monitoring</td>
<td>Enables monitoring if true, and disables monitoring if false.</td>
</tr>
</tbody>
</table>

**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. Equivalent to issuing ALTER TABLE ... MONITORING (or NOMONITORING) individually. It is recommended to enable monitoring if you use gather_database_stats or gather_schema_stats with the GATHER AUTO or GATHER STALE options.

**Syntax**

```sql
DBMS_STATS.ALTER_DATABASE_TABLE_MONITORING (    
    monitoring boolean default TRUE, 
    sysobjs boolean default FALSE);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitoring</td>
<td>Enables monitoring if true, and disables monitoring if false.</td>
</tr>
<tr>
<td>sysobjs</td>
<td>If true, changes monitoring on the dictionary objects.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges.
Oracle8i 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
- Using DBMS_TRACE
- Summary of DBMS_TRACE Subprograms
Requirements, Restrictions, and Constants for DBMS_TRACE

Requirements
This package must be created under SYS.

Restrictions
You cannot use PL/SQL tracing with the multi-threaded server (MTS).

Constants
DBMS_TRACE uses these constants:

```plaintext
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:

```sql
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be
```
Using DBMS_TRACE

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 SYSTEM.

Collecting Trace Data
The PL/SQL features you can trace are described in the script DBMS_PBT.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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET_PLSQL_TRACE Procedure&quot; on page 64-5</td>
<td>Starts tracing in the current session.</td>
</tr>
<tr>
<td>&quot;CLEAR_PLSQL_TRACE Procedure&quot; on page 64-6</td>
<td>Stops trace data dumping in session.</td>
</tr>
<tr>
<td>&quot;PLSQL_TRACE_VERSION Procedure&quot; on page 64-6</td>
<td>Gets the version number of the trace package.</td>
</tr>
</tbody>
</table>

### SET_PLSQL_TRACE Procedure
This procedure enables PL/SQL trace data collection.
CLEAR_PLSQL_TRACE Procedure

Syntax

DBMS_TRACE.SET_PLSQL_TRACE (trace_level INTEGER);

Parameter

Table 64–2 describes the parameter for the DBMS_TRACE.SET_PLSQL_TRACE syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace_level</td>
<td>You must supply one or more of the constants as listed on page 64-2. By summing the constants, you can enable tracing of multiple PL/SQL language features simultaneously. The control constants &quot;trace_pause&quot;, &quot;trace_resume&quot; and &quot;trace_stop&quot; should not be used in combination with other constants. Also see &quot;Collecting Trace Data&quot; on page 64-3 for more information.</td>
</tr>
</tbody>
</table>

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 64–3 describes the parameters for the DBMS_TRACE.PLSQL_TRACE_VERSION syntax.
### Table 64–3  PLSQL_TRACE_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version number of DBMS_TRACE.</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version number of DBMS_TRACE.</td>
</tr>
</tbody>
</table>
PLSQL_TRACE_VERSION Procedure
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>
<thead>
<tr>
<th>Subprogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;READ_ONLY Procedure&quot; on page 65-2</td>
</tr>
<tr>
<td>&quot;READ_WRITE Procedure&quot; on page 65-3</td>
</tr>
<tr>
<td>&quot;ADVISE_ROLLBACK Procedure&quot; on page 65-3</td>
</tr>
<tr>
<td>&quot;ADVISE NOTHING Procedure&quot; on page 65-3</td>
</tr>
<tr>
<td>&quot;ADVISE_COMMIT Procedure&quot; on page 65-3</td>
</tr>
<tr>
<td>&quot;USE_ROLLBACK_SEGMENT Procedure&quot; on page 65-4</td>
</tr>
<tr>
<td>&quot;COMMITS COMMENT Procedure&quot; on page 65-4</td>
</tr>
<tr>
<td>&quot;COMMIT_FORCE Procedure&quot; on page 65-4</td>
</tr>
<tr>
<td>&quot;COMMIT Procedure&quot; on page 65-5</td>
</tr>
<tr>
<td>&quot;SAVEPOINT Procedure&quot; on page 65-5</td>
</tr>
<tr>
<td>&quot;ROLLBACK Procedure&quot; on page 65-6</td>
</tr>
<tr>
<td>&quot;ROLLBACK_SAVEPOINT Procedure&quot; on page 65-6</td>
</tr>
<tr>
<td>&quot;ROLLBACK_FORCE Procedure&quot; on page 65-6</td>
</tr>
<tr>
<td>&quot;BEGIN_DISCRETE_TRANSACTION Procedure&quot; on page 65-7</td>
</tr>
<tr>
<td>&quot;PURGE_MIXED Procedure&quot; on page 65-7</td>
</tr>
<tr>
<td>&quot;PURGE_LOST_DB_ENTRY Procedure&quot; on page 65-8</td>
</tr>
<tr>
<td>&quot;LOCAL TRANSACTION_ID Function&quot; on page 65-10</td>
</tr>
<tr>
<td>&quot;STEP_ID Function&quot; on page 65-11</td>
</tr>
</tbody>
</table>

READ_ONLY Procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION READ ONLY
```
Summary of DBMS_TRANSACTION Subprograms

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb_name</td>
<td>Name of rollback segment to use.</td>
</tr>
</tbody>
</table>

COMMIT_COMMENT Procedure

This procedure is equivalent to following SQL statement:

```
COMMIT COMMENT <text>
```

Syntax

```
DBMS_TRANSACTION.COMMIT_COMMENT ( 
    cmnt VARCHAR2); 
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmnt</td>
<td>Comment to associate with this commit.</td>
</tr>
</tbody>
</table>

COMMIT_FORCE Procedure

This procedure is equivalent to following SQL statement:

```
COMMIT FORCE <text>, <number>" 
```

Syntax

```
DBMS_TRANSACTION.COMMIT_FORCE ( 
```

65-4 Supplied PL/SQL Packages and Types Reference
xid VARCHAR2,
scn VARCHAR2 DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Table 65–4 COMMIT_FORCE Procedure Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>xid</td>
</tr>
<tr>
<td>scn</td>
</tr>
</tbody>
</table>

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);
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 65–6  ROLLBACK_SAVEPOINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>savept</td>
<td>Savpoint identifier.</td>
</tr>
</tbody>
</table>

ROLLBACK_FORCE Procedure

This procedure is equivalent to following SQL statement:

ROLLBACK FORCE <text>

Syntax

DBMS_TRANSACTION.ROLLBACK_FORCE (xid VARCHAR2);
Summary of DBMS_TRANSACTION Subprograms

Parameters

| Table 65–7 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 65–8 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 \texttt{DBA\_2PC\_PENDING} by setting the \texttt{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

\begin{verbatim}
DBMS\_TRANSACTION.PURGE\_MIXED (  
    xid VARCHAR2);
\end{verbatim}

Parameters

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
\textbf{Parameter} & \textbf{Description} \\
\hline
\texttt{xid} & Must be set to the value of the \texttt{LOCAL\_TRAN\_ID} column in the \texttt{DBA\_2PC\_PENDING} table. \\
\hline
\end{tabular}
\caption{\texttt{PURGE\_MIXED} Procedure Parameters}
\end{table}

\textbf{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 recreated before recovery completes, then the entries used to control recovery in \texttt{DBA\_2PC\_PENDING} and associated tables are never removed, and recovery will periodically retry. Procedure \texttt{PURGE\_LOST\_DB\_ENTRY} enables removal of such transactions from the local site.

Syntax

\begin{verbatim}
DBMS\_TRANSACTION.PURGE\_LOST\_DB\_ENTRY (  
    xid VARCHAR2);
\end{verbatim}

\textbf{WARNING:} \texttt{PURGE\_LOST\_DB\_ENTRY} should only be used when the other database is lost or has been recreated. Any other use may leave the other database in an unrecoverable or inconsistent state.

Before automatic recovery runs, the transaction may show up in \texttt{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 recreated, 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 65–10  PURGE_LOST_DB_ENTRY Procedure States**

<table>
<thead>
<tr>
<th>State of Column</th>
<th>State of Global Transaction</th>
<th>State of Local Transaction</th>
<th>Normal DBA Action</th>
<th>Alternative DBA Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collecting</td>
<td>Rolled back</td>
<td>Rolled back</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY</td>
</tr>
<tr>
<td>Committed</td>
<td>Committed</td>
<td>Committed</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY</td>
</tr>
<tr>
<td>Prepared</td>
<td>Unknown</td>
<td>Prepared</td>
<td>None</td>
<td>FORCE COMMIT or ROLLBACK</td>
</tr>
<tr>
<td>Forced commit</td>
<td>Unknown</td>
<td>Committed</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY</td>
</tr>
<tr>
<td>Forced rollback</td>
<td>Unknown</td>
<td>Rolled back</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY</td>
</tr>
<tr>
<td>Forced commit (mixed)</td>
<td>Mixed</td>
<td>Committed</td>
<td>(See Note 2)</td>
<td></td>
</tr>
<tr>
<td>Forced rollback (mixed)</td>
<td>Mixed</td>
<td>Rolled back</td>
<td>(See Note 2)</td>
<td></td>
</tr>
</tbody>
</table>
LOCAL_TRANSACTION_ID Function

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 65–11 PURGE_LOST_DB_ENTRY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.</td>
</tr>
</tbody>
</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

```sql
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (  
    create_transaction BOOLEAN := FALSE)  
RETURN VARCHAR2;
```

Parameters

Table 65–12 LOCAL_TRANSACTION_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_transaction</td>
<td>If true, then start a transaction if one is not currently active.</td>
</tr>
</tbody>
</table>
**STEP_ID Function**

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

**Syntax**

```sql
DBMS_TRANSACTION.STEP_ID
    RETURN NUMBER;
```
The DBMS_TRANSFORM package provides an interface to the message format transformation features of Oracle Advanced Queuing. See the 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 66–1  DBMS_TRANSFORM Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CREATE_TRANSFORMATION Procedure&quot; on page 66-2</td>
<td>Creates a transformation that maps an object of the source type to an object of the destination type</td>
</tr>
<tr>
<td>&quot;MODIFY_TRANSFORMATION Procedure&quot; on page 66-3</td>
<td>Modifies an existing transformation</td>
</tr>
<tr>
<td>&quot;DROP_TRANSFORMATION Procedure&quot; on page 66-4</td>
<td>Drops the given transformation</td>
</tr>
</tbody>
</table>

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 PLSQL function. It must return an object of the target type.

Syntax

```sql
DBMS_TRANSFORM.CREATE_TRANSFORMATION (  
schema VARCHAR2(30),  
name VARCHAR2(30),  
to_schema VARCHAR2(30),  
to_name VARCHAR2(30),  
from_schema VARCHAR2(30),  
from_name VARCHAR2(30),  
transformation VARCHAR2(4000));
```

You can also specify the transformation expression, which returns 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 to not specify a transformation expression and specify a transformation per attribute of the target_type using MODIFY_TRANSFORMATION.
Parameters

Table 66–2 CREATE_TRANSFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies the schema of the transformation</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the transformation.</td>
</tr>
<tr>
<td>from_schema</td>
<td>Specifies the schema of the source type</td>
</tr>
<tr>
<td>from_name</td>
<td>Specifies the source type.</td>
</tr>
<tr>
<td>to_schema</td>
<td>Specifies the target type schema</td>
</tr>
<tr>
<td>to_type</td>
<td>Specifies the target type.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies the transformation expression, returning an object of the target type.</td>
</tr>
</tbody>
</table>

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 PLSQL 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 PLSQL function returning an object of the target type or a constructor expression for the target type.

Syntax

```sql
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (  
  schema VARCHAR2(30),  
  name VARCHAR2(30),  
  attribute_number INTEGER,  
  transformation VARCHAR2(4000));
```

Parameters

Table 66–3 ADD_ATTRIBUTE_MAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies the schema of the transformation</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the transformation.</td>
</tr>
<tr>
<td>attribute_number</td>
<td>Musts be zero for a scalar target type.</td>
</tr>
</tbody>
</table>
DROP_TRANSFORMATION Procedure

This procedure drops the given transformation.

Syntax

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
    schema       VARCHAR2 (30),
    name         VARCHAR2 (30);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transformation_expression</td>
<td>The transformation expression must be a SQL expression or a PLSQL function returning the type of the specified attribute of the target type.</td>
</tr>
</tbody>
</table>
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
- Summary of DBMS_TTS Subprograms
Exceptions

```plsql
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>
<thead>
<tr>
<th>DBMS_TTS Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table 67–1</strong></td>
</tr>
<tr>
<td><strong>Subprogram</strong></td>
</tr>
<tr>
<td>&quot;TRANSPORT_SET_CHECK Procedure&quot; on page 67-2</td>
</tr>
<tr>
<td>&quot;DOWNGRADE Procedure&quot; on page 67-3</td>
</tr>
</tbody>
</table>

**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**

```plsql
DBMS_TTS.TRANSPORT_SET_CHECK (  
  ts_list    IN VARCHAR2,
  incl_constraints IN BOOLEAN DEFAULT,
  full_closure IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 67–2  TRANSPORT_SET_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts_list</td>
<td>List of tablespace, separated by comma.</td>
</tr>
<tr>
<td>incl_constraints</td>
<td>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.)</td>
</tr>
<tr>
<td>full_closure</td>
<td>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.</td>
</tr>
</tbody>
</table>

DOWNGRADE Procedure

This procedure downgrades transportable tablespace related data.

Syntax

DBMS_TTS.DOWNGRADE;
The `DBMS_TYPES` package consists of constants, which represent the built-in and user-defined types. See the Oracle interMedia User’s Guide for a complete discussion of types.

This chapter discusses the following topics:

- Constants for `DBMS_TYPES`
### Constants for DBMS_TYPES

The following table lists the constants in the `DBMS_TYPES` package.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPECODE_DATE</td>
<td>A DATE type</td>
</tr>
<tr>
<td>TYPECODE_NUMBER</td>
<td>A NUMBER type</td>
</tr>
<tr>
<td>TYPECODE_RAW</td>
<td>A RAW type</td>
</tr>
<tr>
<td>TYPECODE_CHAR</td>
<td>A CHAR type</td>
</tr>
<tr>
<td>TYPECODE_VARCHAR2</td>
<td>A VARCHAR2 type</td>
</tr>
<tr>
<td>TYPECODE_VARCHAR</td>
<td>A VARCHAR type</td>
</tr>
<tr>
<td>TYPECODE_MLSLABEL</td>
<td>An MLSLABEL type</td>
</tr>
<tr>
<td>TYPECODE_BLOB</td>
<td>A BLOB type</td>
</tr>
<tr>
<td>TYPECODE_BFILE</td>
<td>A BFILE type</td>
</tr>
<tr>
<td>TYPECODE_CLOB</td>
<td>A CLOB type</td>
</tr>
<tr>
<td>TYPECODE_CFILE</td>
<td>A CFILE type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP</td>
<td>A TIMESTAMP type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP_TZ</td>
<td>A TIMESTAMP_TZ type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP_LTZ</td>
<td>A TIMESTAMP_LTZ type</td>
</tr>
<tr>
<td>TYPECODE_INTERVAL_YM</td>
<td>A INTERVAL_YM type</td>
</tr>
<tr>
<td>TYPECODE_INTERVAL_DS</td>
<td>An INTERVAL_DS type</td>
</tr>
<tr>
<td>TYPECODE_REF</td>
<td>A REF type</td>
</tr>
<tr>
<td>TYPECODE_OBJECT</td>
<td>An OBJECT type</td>
</tr>
<tr>
<td>TYPECODE_VARRAY</td>
<td>A VARRAY collection type</td>
</tr>
<tr>
<td>TYPECODE_TABLE</td>
<td>A nested table collection type</td>
</tr>
<tr>
<td>TYPECODE_NAMEDCOLLECTION</td>
<td>An OPAQUE type</td>
</tr>
<tr>
<td>SUCCESS</td>
<td></td>
</tr>
<tr>
<td>NO_DATA</td>
<td></td>
</tr>
</tbody>
</table>

### Exceptions

- INVALID_PARAMETERS
- INCORRECT_USAGE
- TYPE_MISMATCH
This package provides various utility subprograms.

DBMS.Utility submits a job for each partition. It is the user's 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
- Summary of DBMS.Utility Subprograms
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 via 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 69–1  DBMS_UTILITY Subprograms  (Page 1 of 3)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;COMPILE_SCHEMA Procedure&quot; on page 69-4</td>
<td>Compiles all procedures, functions, packages, and triggers in the specified schema.</td>
</tr>
<tr>
<td>&quot;ANALYZE_SCHEMA Procedure&quot; on page 69-5</td>
<td>Analyzes all the tables, clusters, and indexes in a schema.</td>
</tr>
</tbody>
</table>
### Table 69–1  DBMS_UTILITY Subprograms (Page 2 of 3)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ANALYZE_DATABASE Procedure&quot; on page 69-5</td>
<td>Analyzes all the tables, clusters, and indexes in a database.</td>
</tr>
<tr>
<td>&quot;FORMAT_ERROR_STACK Function&quot; on page 69-6</td>
<td>Formats the current error stack.</td>
</tr>
<tr>
<td>&quot;FORMAT_CALL_STACK Function&quot; on page 69-7</td>
<td>Formats the current call stack.</td>
</tr>
<tr>
<td>&quot;IS_CLUSTER_DATABASE Function&quot; on page 69-7</td>
<td>Finds out if this database is running in cluster database mode.</td>
</tr>
<tr>
<td>GET_TIME Function on page 69-7</td>
<td>Finds out the current time in 100th’s of a second.</td>
</tr>
<tr>
<td>&quot;GET_PARAMETER_VALUE Function&quot; on page 69-8</td>
<td>Gets the value of specified init.ora parameter.</td>
</tr>
<tr>
<td>&quot;NAME_RESOLVE Procedure&quot; on page 69-9</td>
<td>Resolves the given name.</td>
</tr>
<tr>
<td>&quot;NAME_TOKENIZE Procedure&quot; on page 69-11</td>
<td>Calls the parser to parse the given name.</td>
</tr>
<tr>
<td>&quot;COMMA_TO_TABLE Procedure&quot; on page 69-11</td>
<td>Converts a comma-separated list of names into a PL/SQL table of names.</td>
</tr>
<tr>
<td>&quot;TABLE_TO_COMMA Procedure&quot; on page 69-12</td>
<td>Converts a PL/SQL table of names into a comma-separated list of names.</td>
</tr>
<tr>
<td>&quot;PORT_STRING Function&quot; on page 69-12</td>
<td>Returns a string that uniquely identifies the version of Oracle and the operating system.</td>
</tr>
<tr>
<td>&quot;DB_VERSION Procedure&quot; on page 69-13</td>
<td>Returns version information for the database.</td>
</tr>
<tr>
<td>&quot;MAKE_DATA_BLOCK_ADDRESS Function&quot; on page 69-13</td>
<td>Creates a data block address given a file number and a block number.</td>
</tr>
<tr>
<td>&quot;DATA_BLOCK_ADDRESS_FILE Function&quot; on page 69-14</td>
<td>Gets the file number part of a data block address.</td>
</tr>
<tr>
<td>&quot;DATA_BLOCK_ADDRESS_BLOCK Function&quot; on page 69-15</td>
<td>Gets the block number part of a data block address.</td>
</tr>
<tr>
<td>&quot;GET_HASH_VALUE Function&quot; on page 69-15</td>
<td>Computes a hash value for the given string.</td>
</tr>
</tbody>
</table>
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 69–2  COMPILE_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Name of the schema.</td>
</tr>
</tbody>
</table>

Exceptions

Table 69–3  COMPILE_SCHEMA Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>Insufficient privileges for some object in this schema.</td>
</tr>
</tbody>
</table>
ANALYZE_SCHEMA Procedure

This procedure analyzes all the tables, clusters, and indexes in a schema.

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 69–4  ANALYZE_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>method</td>
<td>One of ESTIMATE, COMPUTE or DELETE.</td>
</tr>
<tr>
<td>estimate_rows</td>
<td>Number of rows to estimate.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate.</td>
</tr>
<tr>
<td>method_opt</td>
<td>Method options of the following format: [ FOR TABLE ] [ FOR ALL INDEXED ]</td>
</tr>
<tr>
<td></td>
<td>Columns ] [ SIZE n ] [ FOR ALL INDEXES ]</td>
</tr>
</tbody>
</table>

Exceptions

Table 69–5  ANALYZE_SCHEMA Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>Insufficient privileges for some object in this schema.</td>
</tr>
</tbody>
</table>

ANALYZE_DATABASE Procedure

This procedure analyzes all the tables, clusters, and indexes in a 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

```sql
DBMS_UTILITY FORMAT_ERROR_STACK

RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>method</code></td>
<td>One of <code>ESTIMATE</code>, <code>COMPUTE</code> or <code>DELETE</code>.</td>
</tr>
<tr>
<td></td>
<td>If <code>ESTIMATE</code>, then either <code>estimate_rows</code> or <code>estimate_percent</code> must be non-zero.</td>
</tr>
<tr>
<td><code>estimate_rows</code></td>
<td>Number of rows to estimate.</td>
</tr>
<tr>
<td><code>estimate_percent</code></td>
<td>Percentage of rows to estimate.</td>
</tr>
<tr>
<td><code>method_opt</code></td>
<td>Method options of the following format:</td>
</tr>
<tr>
<td></td>
<td>[ FOR TABLE ]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL [INDEXED] COLUMNS] [SIZE n]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL INDEXES ]</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>Insufficient privileges for some object in this database.</td>
</tr>
</tbody>
</table>

ANALYZE_DATABASE Procedure

Syntax

```sql
DBMS_UTILITY.ANALYZE_DATABASE (method VARCHAR2,
                               estimate_rows NUMBER DEFAULT NULL,
                               estimate_percent NUMBER DEFAULT NULL,
                               method_opt VARCHAR2 DEFAULT NULL);
```

Parameters

Table 69–6 ANALYZE_DATABASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>method</code></td>
<td>One of <code>ESTIMATE</code>, <code>COMPUTE</code> or <code>DELETE</code>.</td>
</tr>
<tr>
<td></td>
<td>If <code>ESTIMATE</code>, then either <code>estimate_rows</code> or <code>estimate_percent</code> must be non-zero.</td>
</tr>
<tr>
<td><code>estimate_rows</code></td>
<td>Number of rows to estimate.</td>
</tr>
<tr>
<td><code>estimate_percent</code></td>
<td>Percentage of rows to estimate.</td>
</tr>
<tr>
<td><code>method_opt</code></td>
<td>Method options of the following format:</td>
</tr>
<tr>
<td></td>
<td>[ FOR TABLE ]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL [INDEXED] COLUMNS] [SIZE n]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL INDEXES ]</td>
</tr>
</tbody>
</table>

Table 69–7 ANALYZE_DATABASE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>Insufficient privileges for some object in this database.</td>
</tr>
</tbody>
</table>
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**

```sql
DBMS_UTILITY.FORMAT_CALL_STACK
RETURN VARCHAR2;
```

**Pragmas**

```sql
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**

```sql
DBMS_UTILITY.IS_CLUSTER_DATABASE
RETURN BOOLEAN;
```

Returns

This 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**

```sql
DBMS_UTILITY.GET_TIME
RETURN NUMBER;
```
GET_PARAMETER_VALUE Function

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parnam</td>
<td>Parameter name.</td>
</tr>
<tr>
<td>intval</td>
<td>Value of an integer parameter or the value length of a string parameter.</td>
</tr>
<tr>
<td>strval</td>
<td>Value of a string parameter.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partyp</td>
<td>Parameter type:</td>
</tr>
<tr>
<td></td>
<td>0 if parameter is an integer/boolean parameter</td>
</tr>
<tr>
<td></td>
<td>1 if parameter is a string/file parameter</td>
</tr>
</tbody>
</table>

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);
```
NAME_RESOLVE Procedure

Parameters

Table 69–10  NAME_RESOLVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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 multi-byte).</td>
</tr>
<tr>
<td>context</td>
<td>Must be an integer between 0 and 8.</td>
</tr>
<tr>
<td>schema</td>
<td>Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.</td>
</tr>
<tr>
<td>part1</td>
<td>First part of the name. The type of this name is specified part1_type (synonym, procedure or package).</td>
</tr>
<tr>
<td>part2</td>
<td>If this is non-NULL, then this is a procedure name within the package indicated by part1.</td>
</tr>
<tr>
<td>dblink</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 DBMSUTILITY.NAME_RESOLVE procedure on this remote node. |

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

```sql
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-separated 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

```sql
DBMS_UTILITY.COMMA_TO_TABLE (  
    list IN VARCHAR2,  
    tablen OUT BINARY_INTEGER,  
    tab OUT UNCL_ARRAY);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Comma separated list of tables.</td>
</tr>
<tr>
<td>tablen</td>
<td>Number of tables in the PL/SQL table.</td>
</tr>
<tr>
<td>tab</td>
<td>PL/SQL table which contains list of table names.</td>
</tr>
</tbody>
</table>
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-separated list: Anything other than a comma-separated list is rejected. Commas inside double quotes do not count.

Entries in the comma-separated list cannot include multi-byte 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-separated 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 69–12 TABLE_TO_COMMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>PL/SQL table which contains list of table names.</td>
</tr>
<tr>
<td>tablen</td>
<td>Number of tables in the PL/SQL table.</td>
</tr>
<tr>
<td>list</td>
<td>Comma separated list of tables.</td>
</tr>
</tbody>
</table>

Returns

Returns a comma-separated list and the number of elements found in the table (n).

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**

```plaintext
DBMS_UTILITY.PORT_STRING
    RETURN VARCHAR2;
```

**Pragmas**

```plaintext
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

**DB_VERSION Procedure**

This procedure returns version information for the database.

**Syntax**

```plaintext
DBMS_UTILITY.DB_VERSION (
    version OUT VARCHAR2,
    compatibility OUT VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>A string which represents the internal software version of the database (e.g., 7.1.0.0.0). The length of this string is variable and is determined by the database version.</td>
</tr>
<tr>
<td>compatibility</td>
<td>The compatibility setting of the database determined by the &quot;compatible&quot; init.ora parameter. If the parameter is not specified in the init.ora file, then NULL is returned.</td>
</tr>
</tbody>
</table>

**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.
DATA_BLOCK_ADDRESS_FILE Function

Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (
    file NUMBER,
    block NUMBER)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File that contains the block.</td>
</tr>
<tr>
<td>block</td>
<td>Offset of the block within the file in terms of block increments.</td>
</tr>
</tbody>
</table>

Pragmas

```
pragma restrict_references(make_data_block_address, WNDS, RNDS, WNPS, RNPS);  
```

Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>Data block address.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>Data block address.</td>
</tr>
</tbody>
</table>
Summary of DBMSUTILITY Subprograms

Pragmas

Pragma restrict_references(data_block_address_file, WNDS, RNDS, WNPS, RNPS);

Returns

Table 69–17 DATA_BLOCK_ADDRESS_FILE Function Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File that contains the block.</td>
</tr>
</tbody>
</table>

DATA_BLOCK_ADDRESS_BLOCK Function

This function gets the block number part of a data block address.

Syntax

DBMSUTILITY.DATA_BLOCK_ADDRESS_BLOCK (dba NUMBER) RETURN NUMBER;

Parameters

Table 69–18 DATA_BLOCK_ADDRESS_BLOCK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>Data block address.</td>
</tr>
</tbody>
</table>

Pragmas

Pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);

Returns

Table 69–19 DATA_BLOCK_ADDRESS_BLOCK Function Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>Block offset of the block.</td>
</tr>
</tbody>
</table>

GET_HASH_VALUE Function

This function computes a hash value for the given string.
ANALYZE_PART_OBJECT Procedure

Syntax

```sql
DBMS_UTILITY.GET_HASH_VALUE (
    name       VARCHAR2,
    base       NUMBER,
    hash_size  NUMBER)
RETURN NUMBER;
```

Parameters

```sql
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String to be hashed.</td>
</tr>
<tr>
<td>base</td>
<td>Base value for the returned hash value to start at.</td>
</tr>
<tr>
<td>hash_size</td>
<td>Desired size of the hash table.</td>
</tr>
</tbody>
</table>
```

Pragmas

```sql
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:

```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

```sql
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 69–21 ANALYZE_PART_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Schema of the object_name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of object to be analyzed, must be partitioned.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of object, must be T (table) or I (index).</td>
</tr>
<tr>
<td>command_type</td>
<td>Must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>C (compute statistics)</td>
</tr>
<tr>
<td></td>
<td>E (estimate statistics)</td>
</tr>
<tr>
<td></td>
<td>D (delete statistics)</td>
</tr>
<tr>
<td></td>
<td>V (validate structure)</td>
</tr>
<tr>
<td>command_opt</td>
<td>Other options for the command type.</td>
</tr>
<tr>
<td></td>
<td>For C, E it can be FOR table, FOR all LOCAL indexes,</td>
</tr>
<tr>
<td></td>
<td>FOR all columns or combination of some of the ‘for’</td>
</tr>
<tr>
<td></td>
<td>options of analyze statistics (table). For V, it</td>
</tr>
<tr>
<td></td>
<td>can be CASCADE when object_type is T.</td>
</tr>
<tr>
<td>sample_clause</td>
<td>The sample clause to use when command_type is ‘E’.</td>
</tr>
</tbody>
</table>

EXEC_DDL_STATEMENT Procedure

This procedure executes the DDL statement in parse_string.

Syntax

```
DBMSUTILITY.EXEC_DDL_STATEMENT (            
   parse_string IN VARCHAR2);            
```

Parameters

Table 69–22 EXEC_DDL_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse_string</td>
<td>DDL statement to be executed.</td>
</tr>
</tbody>
</table>
CURRENT_INSTANCE Function

This function returns the current connected instance number. It returns NULL when connected instance is down.

Syntax

```sql
DBMS_UTILITY.CURRENT_INSTANCE
RETURN NUMBER;
```

ACTIVE_INSTANCES Procedure

Syntax

```sql
DBMS_UTILITY.ACTIVE_INSTANCE (  
    instance_table OUT INSTANCE_TABLE,  
    instance_count OUT NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_table</td>
<td>Contains a list of the active instance numbers and names.</td>
</tr>
<tr>
<td></td>
<td>When no instance is up, the list is empty.</td>
</tr>
<tr>
<td>instance_count</td>
<td>Number of active instances.</td>
</tr>
</tbody>
</table>
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. For detailed conceptual and usage information about Workspace Manager, including descriptions of the types of procedures, see Oracle9i Application Developer’s Guide - Workspace Manager. That manual also includes the detailed reference information found in this chapter.

This chapter discusses the following topics:

- Summary of DBMS_WM Subprograms
Summary of DBMS_WM Subprograms

Table 70–1  DBMS_WM Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;AlterSavepoint Procedure&quot; on page 70-5</td>
<td>Modifies the description of a savepoint.</td>
</tr>
<tr>
<td>&quot;AlterWorkspace Procedure&quot; on page 70-6</td>
<td>Modifies the description of a workspace.</td>
</tr>
<tr>
<td>&quot;BeginResolve Procedure&quot; on page 70-7</td>
<td>Starts a conflict resolution session.</td>
</tr>
<tr>
<td>&quot;CommitResolve Procedure&quot; on page 70-8</td>
<td>Ends a conflict resolution session and saves (makes permanent) any changes</td>
</tr>
<tr>
<td>&quot;CompressWorkspace Procedure&quot; on page 70-9</td>
<td>in the workspace since BeginResolve was executed.</td>
</tr>
<tr>
<td>&quot;CompressWorkspaceTree Procedure&quot; on page 70-11</td>
<td>Deletes explicit savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace.</td>
</tr>
<tr>
<td>&quot;CopyForUpdate Procedure&quot; on page 70-13</td>
<td>Allows LOB columns (BLOB or CLOB) in version-enabled tables to be modified.</td>
</tr>
<tr>
<td>&quot;CreateSavepoint Procedure&quot; on page 70-14</td>
<td>Creates a savepoint for the current version.</td>
</tr>
<tr>
<td>&quot;CreateWorkspace Procedure&quot; on page 70-16</td>
<td>Creates a new workspace in the database.</td>
</tr>
<tr>
<td>&quot;DeleteSavepoint Procedure&quot; on page 70-17</td>
<td>Deletes a savepoint.</td>
</tr>
<tr>
<td>&quot;DisableVersioning Procedure&quot; on page 70-19</td>
<td>Deletes all support structures that were created to enable the table to support versioned rows.</td>
</tr>
<tr>
<td>&quot;EnableVersioning Procedure&quot; on page 70-21</td>
<td>Creates the necessary structures to enable the table to support multiple versions of rows.</td>
</tr>
<tr>
<td>&quot;FreezeWorkspace Procedure&quot; on page 70-22</td>
<td>Disables changes in a workspace and prevents subsequent sessions from entering the workspace.</td>
</tr>
<tr>
<td>&quot;GetConflictWorkspace Function&quot; on page 70-24</td>
<td>Returns the name of the workspace on which the session has performed the SetConflictWorkspace procedure.</td>
</tr>
</tbody>
</table>
**Table 70–1  DBMS_WM Subprograms (Cont.)**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GetDiffVersions Function&quot; on page 70-24</td>
<td>Returns the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions operation.</td>
</tr>
<tr>
<td>&quot;GetLockMode Function&quot; on page 70-25</td>
<td>Returns the locking mode, which determines whether or not access is enabled to versioned rows and corresponding rows in the parent workspace.</td>
</tr>
<tr>
<td>&quot;GetMultiWorkspaces Function&quot; on page 70-26</td>
<td>Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.</td>
</tr>
<tr>
<td>&quot;GetOpContext Function&quot; on page 70-26</td>
<td>Returns the context of the current operation.</td>
</tr>
<tr>
<td>&quot;GetPrivs Function&quot; on page 70-27</td>
<td>Returns a comma-separated list of all privileges that the current user has for the specified workspace.</td>
</tr>
<tr>
<td>&quot;GetWorkspace Function&quot; on page 70-28</td>
<td>Returns the current workspace for the session.</td>
</tr>
<tr>
<td>&quot;GotoDate Procedure&quot; on page 70-29</td>
<td>Goes to a point at or near the specified date and time in the current workspace.</td>
</tr>
<tr>
<td>&quot;GotoSavepoint Procedure&quot; on page 70-30</td>
<td>Goes to the specified savepoint in the current workspace.</td>
</tr>
<tr>
<td>&quot;GotoWorkspace Procedure&quot; on page 70-31</td>
<td>Moves the current session to the specified workspace.</td>
</tr>
<tr>
<td>&quot;GrantSystemPriv Procedure&quot; on page 70-32</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;GrantWorkspacePriv Procedure&quot; on page 70-34</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;IsWorkspaceOccupied Function&quot; on page 70-36</td>
<td>Checks whether or not a workspace has any active sessions.</td>
</tr>
<tr>
<td>&quot;LockRows Procedure&quot; on page 70-37</td>
<td>Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.</td>
</tr>
<tr>
<td>&quot;MergeTable Procedure&quot; on page 70-38</td>
<td>Applies changes to a table (all rows or as specified in the WHERE clause) in a workspace to its parent workspace.</td>
</tr>
<tr>
<td>&quot;MergeWorkspace Procedure&quot; on page 70-40</td>
<td>Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.</td>
</tr>
</tbody>
</table>
### Table 70–1  DBMS_WM Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;RefreshTable Procedure&quot; on page 70-41</td>
<td>Applies to a workspace all changes made to a table (all rows or as specified in the \texttt{WHERE} clause) in its parent workspace.</td>
</tr>
<tr>
<td>&quot;RefreshWorkspace Procedure&quot; on page 70-43</td>
<td>Applies to a workspace all changes made in its parent workspace.</td>
</tr>
<tr>
<td>&quot;RemoveWorkspace Procedure&quot; on page 70-44</td>
<td>Rolls back the data in the workspace and removes all support structures created for the workspace. The workspace ceases to exist.</td>
</tr>
<tr>
<td>&quot;RemoveWorkspaceTree Procedure&quot; on page 70-45</td>
<td>Removes the specified workspace and all its descendant workspaces. The data in the workspaces is rolled back and the workspace structure is removed.</td>
</tr>
<tr>
<td>&quot;ResolveConflicts Procedure&quot; on page 70-46</td>
<td>Resolves conflicts between workspaces.</td>
</tr>
<tr>
<td>&quot;RevokeSystemPriv Procedure&quot; on page 70-48</td>
<td>Revokes (removes) system-level privileges from users and roles.</td>
</tr>
<tr>
<td>&quot;RevokeWorkspacePriv Procedure&quot; on page 70-50</td>
<td>Revokes (removes) workspace-level privileges from users and roles for a specified workspace.</td>
</tr>
<tr>
<td>&quot;RollbackResolve Procedure&quot; on page 70-51</td>
<td>Quits a conflict resolution session and discards all changes in the workspace since BeginResolve was executed.</td>
</tr>
<tr>
<td>&quot;RollbackTable Procedure&quot; on page 70-52</td>
<td>Discards all changes made in the workspace to a specified table (all rows or as specified in the \texttt{WHERE} clause).</td>
</tr>
<tr>
<td>&quot;RollbackToSP Procedure&quot; on page 70-53</td>
<td>Discards all changes made after a specified savepoint in the workspace to all tables.</td>
</tr>
<tr>
<td>&quot;RollbackWorkspace Procedure&quot; on page 70-55</td>
<td>Discards all changes made in the workspace to all tables.</td>
</tr>
<tr>
<td>&quot;SetConflictWorkspace Procedure&quot; on page 70-56</td>
<td>Determine whether or not conflicts exist between a workspace and its parent.</td>
</tr>
<tr>
<td>&quot;SetDiffVersions Procedure&quot; on page 70-57</td>
<td>Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It creates rows in the differences views describing these differences.</td>
</tr>
<tr>
<td>&quot;SetLockingOFF Procedure&quot; on page 70-59</td>
<td>Enables access to versioned rows and to corresponding rows in the parent workspace.</td>
</tr>
<tr>
<td>&quot;SetLockingON Procedure&quot; on page 70-60</td>
<td>Controls access to versioned rows and to corresponding rows in the previous version.</td>
</tr>
<tr>
<td>&quot;SetMultiWorkspaces Procedure&quot; on page 70-61</td>
<td>Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.</td>
</tr>
</tbody>
</table>
AlterSavepoint Procedure

This procedure modifies the description of a savepoint.

Syntax

```sql
DBMS_WM.AlterSavepoint(
    workspace IN VARCHAR2,
    sp_name IN VARCHAR2,
    sp_description IN VARCHAR2);
```
AlterWorkspace Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace in which the savepoint was created. The name is case sensitive.</td>
</tr>
<tr>
<td>sp_name</td>
<td>Name of the savepoint. The name is case sensitive.</td>
</tr>
<tr>
<td>sp_description</td>
<td>Description of the savepoint.</td>
</tr>
</tbody>
</table>

Usage Notes

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.

```sql
EXECUTE DBMS_WM.AlterSavepoint ('NEWWORKSPACE', 'SP1', 'First set of changes for scenario');
```

AlterWorkspace Procedure

This procedure modifies the description of a workspace.

Syntax

```sql
DBMS_WM.AlterWorkspace(
    workspace IN VARCHAR2,
    workspace_description IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>workspace_description</td>
<td>Description of the workspace.</td>
</tr>
</tbody>
</table>
Usage Notes

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');

BeginResolve Procedure

This procedure starts a conflict resolution session.

Syntax

DBMS_WM.BeginResolve(
    workspace IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure starts a conflict resolution session. While this procedure is executing, the workspace is frozen in 1WRITER mode.

After calling this procedure, you can execute ResolveConflicts Procedure as needed for various tables that have conflicts, and then call either 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 regular transactions in workspace.
- The user executing BeginResolve Procedure does not have the privilege to access workspace and its parent workspace.
CommitResolve Procedure

Examples

The following example starts a conflict resolution session in Workspace1.

```sql
EXECUTE DBMS_WM.BeginResolve ('Workspace1');
```

CommitResolve Procedure

This procedure ends a conflict resolution session and saves (makes permanent) any changes in the workspace since BeginResolve Procedure was executed.

Syntax

```sql
DBMS_WM.CommitResolve(
    workspace IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure ends the current conflict resolution session (started by BeginResolve Procedure), and saves all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with 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 regular 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.

```sql
EXECUTE DBMS_WM.CommitResolve ('Workspace1');
```
CompressWorkspace Procedure

This procedure deletes explicit savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace.

Syntax

```
DBMS_WM.CompressWorkspace(
    workspace IN VARCHAR2,
    [compress_view_wo_overwrite IN BOOLEAN DEFAULT FALSE],
    [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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>compress_view_wo_overwrite</td>
<td>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 (the default) 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.)</td>
</tr>
</tbody>
</table>
CompressWorkspace Procedure

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 READ_ONLY mode.

A workspace cannot be compressed if there are any sessions with an open regular transaction, or if any user has executed a GotoDate Procedure operation or a GotoSavepoint Procedure operation specifying a savepoint in the workspace.

Table 70–6  CompressWorkspace Procedure Parameters (Cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstSP</td>
<td>First explicit savepoint. Savepoint names are case sensitive. If only workspace and firstSP are specified, all explicit savepoints between workspace creation and firstSP (but not including firstSP) are deleted. If workspace, firstSP, and secondSP are specified, all explicit savepoints from firstSP (and including firstSP if it is an explicit savepoint) to secondSP (but not including secondSP) are deleted. If only workspace is specified (no savepoints), all explicit savepoints in the workspace are deleted.</td>
</tr>
<tr>
<td>secondSP</td>
<td>Second explicit savepoint. All explicit savepoints from firstSP (and including firstSP if it is an explicit savepoint) to secondSP (but not including secondSP) are deleted. Savepoint names are case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>
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`.

```sql
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE');
```

The following example compresses `NEWWORKSPACE`, deleting all explicit savepoints between the creation of the workspace and the savepoint `SP1`.

```sql
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`.

```sql
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.

```sql
EXECUTE DBMS_WM.CompressWorkspace ('B_focus_1', NULL, NULL, FALSE);
```

### CompressWorkspaceTree Procedure

This procedure deletes explicit 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.

### Syntax

```sql
DBMS_WM.CompressWorkspaceTree(
    workspace IN VARCHAR2,
    [compress_view_wo_overwrite IN BOOLEAN DEFAULT FALSE],
    [auto_commit IN BOOLEAN DEFAULT TRUE]);
```
CompressWorkspaceTree Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>compress_view_wo_overwrite</td>
<td>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.)</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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, you will not need to go to or roll back to any of the savepoints).

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.

A workspace cannot be compressed if there are any sessions with an open regular 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 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.

```sql
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.

```sql
EXECUTE DBMS_WM.CompressWorkspaceTree ('B_focus_1', NULL, FALSE);
```

**CopyForUpdate Procedure**

This 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**

```sql
DBMS_WM.CopyForUpdate(
    table_name IN VARCHAR2,
    [, where_clause IN VARCHAR2 DEFAULT '']);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table containing one or more LOB columns. The name is not case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>The WHERE clause (excluding the WHERE keyword) identifying the rows affected. Example: 'department_id = 20' The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are affected.</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure is intended for use only with version-enabled tables containing one or more LOB columns. The CopyForUpdate 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.)

See the following example.

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, i.e.,
     if the row with doc_id = 1 has not been versioned in the
     current version */
  vm.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

This 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace in which to create the savepoint. The name is case sensitive.</td>
</tr>
<tr>
<td>savepoint_name</td>
<td>Name of the savepoint to be created. The name is case sensitive.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the savepoint to be created.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
</tbody>
</table>

TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes.

FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.

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; a quiet point is not required.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode.

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 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

This procedure creates a new workspace in the database.

Syntax

```sql
DBMS_WM.CreateWorkspace(
    workspace IN VARCHAR2,
    [, isrefreshed IN BOOLEAN DEFAULT FALSE]
    [, description IN VARCHAR2 DEFAULT NULL]
    [, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

or

```sql
DBMS_WM.CreateWorkspace(
    workspace IN VARCHAR2,
    isrefreshed IN BOOLEAN
    [, description IN VARCHAR2 DEFAULT NULL]
    [, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive, and it must be unique (no other workspace of the same name).</td>
</tr>
<tr>
<td>isrefreshed</td>
<td>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. A continually refreshed workspace must be created as a child of the LIVE workspace. 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.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the workspace.</td>
</tr>
</tbody>
</table>
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.

This procedure does not implicitly go to the workspace created. To go to the workspace, use the GotoWorkspace Procedure.

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.

```sql
EXECUTE DBMS_WM.CreateWorkspace ('NEWWORKSPACE');
```

DeleteSavepoint Procedure

This procedure deletes a savepoint.
DeleteSavepoint Procedure

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace in which the savepoint was created. The name is case sensitive.</td>
</tr>
<tr>
<td>savepoint_name</td>
<td>Name of the savepoint to be deleted. The name is case sensitive.</td>
</tr>
<tr>
<td>compress_view_wo_overwrite</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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.

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 regular 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:

- The user is not in the latest version in the workspace (for example, if the user has called GotoDate Procedure).
- workspace does not exist.
- savepoint_name does not exist.
- 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

This 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)

Parameters

Table 70–12  DisableVersioning Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table. The name is not case sensitive.</td>
</tr>
</tbody>
</table>
DisableVersioning Procedure

Table 70–12 DisableVersioning Procedure Parameters (Cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE forces all data in workspaces other than LIVE to be discarded before versioning is disabled.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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.)

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.

Examples

The following example disables the EMPLOYEE table for versioning.

EXECUTE DBMS_WM.DisableVersioning ('employee');
EnableVersioning Procedure

This procedure creates the necessary structures to enable the table to support multiple versions of rows.

Syntax

```sql
DBMS_WM.EnableVersioning(
    table_name IN VARCHAR2,
    hist IN VARCHAR2 DEFAULT 'NONE');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table. The length of a table name must not exceed 25 characters. The name is not case sensitive.</td>
</tr>
<tr>
<td>hist</td>
<td>History option, for tracking modifications to table_name. Must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>NONE: No modifications to the table are tracked. (This is the default.)</td>
</tr>
<tr>
<td></td>
<td>VIEW_W_OVERWRITE: The with overwrite (W_OVERWRITE) option: A view named &lt;table_name&gt;_HIST 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 &lt;table_name&gt;_HIST view contains only the time of the most recent update.)</td>
</tr>
<tr>
<td></td>
<td>VIEW_WO_OVERWRITE: The without overwrite (WO_OVERWRITE) option: A view named &lt;table_name&gt;_HIST 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.</td>
</tr>
</tbody>
</table>

Usage Notes

The table that is being version-enabled must have a primary key defined.

Only the owner of a table 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 the table is already version-enabled.

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 Procedure`. However, the `VIEW_WO_OVERWRITE hist` 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.

Current notes and restrictions include:

- 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 are not allowed on version-enabled tables.
- 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.

**Examples**

The following example enables versioning on the `EMPLOYEE` table.

```sql
EXECUTE DBMS_WM.EnableVersioning('employee');
```

**FreezeWorkspace Procedure**

This procedure disables changes in a workspace and prevents subsequent sessions from entering the workspace.

**Syntax**

```sql
DBMS_WM.FreezeWorkspace(
    workspace IN VARCHAR2
)
```
Summary of DBMS_WM Subprograms

```
[, freezemode  IN VARCHAR2 DEFAULT 'NO_ACCESS']
[, freezewriter IN VARCHAR2 DEFAULT NULL]
[, force      IN BOOLEAN DEFAULT FALSE]);
```

Parameters

**Table 70–14 FreezeWorkspace Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>
| 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).  
  - 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

The operation fails if any sessions are active in workspace (unless force is TRUE) and freezemode is NO_ACCESS.

If freezemode is READ_ONLY or 1WRITER, the workspace cannot be frozen if there is an active regular transaction.
Only the owner of the workspace or a user with **WM_ADMIN_ROLE** can freeze a workspace. There are no specific privileges associated with freezing a 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');
```

**GetConflictWorkspace Function**

This function returns the name of the workspace on which the session has performed the **SetConflictWorkspace Procedure**.

**Syntax**

```
DBMS_WM.GetConflictWorkspace
RETURN VARCHAR2;
```

**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;
```

**GetDiffVersions Function**

This function returns the names of the (workspace, savepoint) pairs on which the session has performed the **SetDiffVersions Procedure** operation.
Summary of DBMS_WM Subprograms

**Syntax**

DBMS_WM.GetDiffVersions
RETURN VARCHAR2;

**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**

This function returns the locking mode, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.

**Syntax**

DBMS_WM.GetLockMode
RETURN VARCHAR2;

**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.
GetMultiWorkspaces Function

Examples

The following example displays the locking mode in effect for the session.

```
SELECT DBMS_WM.GetLockMode FROM DUAL;
```

```
GETLOCKMODE
C
```

GetMultiWorkspaces Function

This function returns the names of workspaces visible in the multiworkspace views for version-enabled tables.

Syntax

```
DBMS_WM.GetMultiWorkspaces
RETURN VARCHAR2;
```

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 GetMultiWorkspaces Procedure.

Examples

The following example displays the names of workspaces visible in the multiworkspace views.

```
SELECT DBMS_WM.GetMultiWorkspaces FROM DUAL;
```

GetOpContext Function

This function returns the context of the current operation.
Syntax

```sql
DBMS_WM.GetOpContext (  
    RETURN VARCHAR2;
)
```

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 `remove_workspace` as `TRUE` or a `MergeTable Procedure` call with `remove_data` as `TRUE`.
- **MERGE_NOREMOVE**: The current operation was initiated by a `MergeWorkspace Procedure` call with `remove_workspace` as `FALSE` or a `MergeTable Procedure` call with `remove_data` as `FALSE`.

Examples

The following example displays the context of the current operation.

```sql
SELECT DBMS_WM.GetOpContext FROM DUAL;
```

GetPrivs Function

This function returns a comma-separated list of all privileges that the current user has for the specified workspace.

Syntax

```sql
DBMS_WM.GetPrivs (  
    workspace  VARCHAR2)  
    RETURN VARCHAR2;
)
GetWorkspace Function

Parameters

**Table 70-15 GetPrivs Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace for which to return the list of privileges. The name is case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

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.

```sql
SELECT DBMS_WM.GetPrivs('B_focus_2') FROM DUAL;
```

```
DBMS_WM.GETPRIVS('B_FOCUS_2')
--------------------------------------------------------------------------------
ACCESS, MERGE, CREATE, REMOVE, ROLLBACK
```

GetWorkspace Function

This function returns the current workspace for the session.

Syntax

```sql
DBMS_WM.GetWorkspace ( 
 RETURN VARCHAR2;
```

Examples

The following example displays the workspace that the current user is in.

```sql
SELECT DBMS_WM.GetWorkspace FROM DUAL;
```

```
GETWORKSPACE
--------------------------------------------------------------------------------
B_focus_2
```
GotoDate Procedure

This procedure goes to a point at or near the specified date and time in the current workspace.

Syntax

```sql
DBMS_WM.GotoDate(
    in_date IN DATE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_date</td>
<td>Date and time for the read-only view of the workspace. (See the Usage Notes for details.)</td>
</tr>
</tbody>
</table>

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 modifications, 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 contents can vary depending on when updates were performed and if or when savepoints were created. The view reflects the data values in effect at `in_date` except for rows that have been modified both (1) between `in_date` and the most recent savepoint before `in_date` and (2) between `in_date` and the next savepoint after `in_date`; for these rows the view reflects the date in effect at the most recent savepoint before `in_date`. Therefore, be careful if you use this procedure when the VIEW_W_OVERWRITE option is enabled.
- 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 effects of the VIEW_W_OVERWRITE and VIEW_WO_OVERWRITE settings. Assume the following sequence of events:

1. The MANAGER_NAME value in a row is Adams.
GotoSavepoint Procedure

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_W_OVERWRITE, the MANAGER_NAME value after step 7 is Adams.
- 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 30-Jun-2000, depending on the history option currently in effect.

EXECUTE DBMS_WM.GotoDate ('30-JUN-00');

GotoSavepoint Procedure

This procedure goes to the specified savepoint in the current workspace.

Syntax

```
DBMS_WM.GotoSavePoint(
    [savepoint_name IN VARCHAR2 DEFAULT 'LATEST']);
```
Parameters

Table 70–17  GotoSavepoint Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>savepoint_name</td>
<td>Name of the savepoint. The name is case sensitive. If savepoint_name is not specified, the default is LATEST.</td>
</tr>
</tbody>
</table>

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 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 GotoSavepoint with a null parameter to go to the currently active version in the workspace. (This achieves the same result as calling 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

This procedure moves the current session to the specified workspace.

Syntax

```
DBMS_WM.GotoWorkspace(   workspace  IN VARCHAR2);
```
GrantSystemPriv Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

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 to new users (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.

```sql
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.

```sql
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
```

GrantSystemPriv Procedure

This procedure 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.

Syntax

```sql
DBMS_WM.GrantSystemPriv(
    priv_types IN VARCHAR2,
```

70-32  Supplied PL/SQL Packages and Types Reference
Parameters

Table 70–19  GrantSystemPriv Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priv_types</td>
<td>A string of one or more keywords representing 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.</td>
</tr>
<tr>
<td>grantee</td>
<td>Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.</td>
</tr>
<tr>
<td>grant_option</td>
<td>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.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes.</td>
</tr>
<tr>
<td></td>
<td>FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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

This procedure 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.

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 70–20  GrantWorkspacePriv Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priv_types</td>
<td>A string of one or more keywords representing privileges. Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.</td>
</tr>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>
Usage Notes

Contrast this procedure with 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**.

### Table 70–20 GrantWorkspacePriv Procedure Parameters (Cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>grantee</strong></td>
<td>Name of the user (can be the <strong>PUBLIC</strong> user group) or role to which to grant <strong>priv_types</strong>.</td>
</tr>
<tr>
<td><strong>grant_option</strong></td>
<td>Specify <strong>YES</strong> to enable the grant option for <strong>grantee</strong>, or <strong>NO</strong> (the default) to disable the grant option for <strong>grantee</strong>. The grant option allows <strong>grantee</strong> to grant the privileges specified in <strong>priv_types</strong> on the workspace specified in <strong>workspace</strong> to other users and roles.</td>
</tr>
<tr>
<td><strong>auto_commit</strong></td>
<td>A boolean value (<strong>TRUE</strong> or <strong>FALSE</strong>). <strong>TRUE</strong> (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. <strong>FALSE</strong> causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>
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

This function checks whether or not a workspace has any active sessions.

Syntax

```
DBMS_WM.IsWorkspaceOccupied(
    workspace IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

This function returns YES if the workspace has any active sessions and 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 active in the B_focus_2 workspace.

```
SELECT DBMS_WM.IsWorkspaceOccupied('B_focus_2') FROM DUAL;
```

```
DBMS_WM.ISWORKSPACEOCCUPIED('B_FOCUS_2')
--------------------------------------------------------------------------------
YES
```

70-36  Supplied PL/SQL Packages and Types Reference
LockRows Procedure

This procedure controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Syntax

```sql
DBMS_WM.LockRows(
    workspace IN VARCHAR2,
    table_name IN VARCHAR2
[, where_clause IN VARCHAR2 DEFAULT '']
[, lock_mode IN VARCHAR2 DEFAULT 'E'])
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>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.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table in which rows are to be locked. The name is not case sensitive.</td>
</tr>
</tbody>
</table>
| 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.
MergeTable 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

This 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>table_id</td>
<td>Name of the table containing rows to be merged into the parent workspace. The name is not case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>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.</td>
</tr>
<tr>
<td>create_savepoint</td>
<td>A boolean value (TRUE or FALSE). TRUE creates an implicit savepoint in the parent workspace before the merge operation. FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.</td>
</tr>
</tbody>
</table>
Summary of DBMS_WM Subprograms

**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 regular transaction affecting the table.

An exception is raised if the user does not have access to `table_id`, or 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

This 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>create_savepoint</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE creates an implicit savepoint in the parent workspace before the merge operation.</td>
</tr>
<tr>
<td></td>
<td>FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.</td>
</tr>
<tr>
<td>remove_workspace</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE removes workspace after the merge operation.</td>
</tr>
<tr>
<td></td>
<td>FALSE (the default) does not remove workspace after the merge operation; the workspace continues to exist.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes.</td>
</tr>
<tr>
<td></td>
<td>FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>
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.

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');

RefreshTable Procedure

This 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

```sql
DBMS_WM.RefreshTable(
    workspace IN VARCHAR2,
    table_id IN VARCHAR2,
    [, where_clause IN VARCHAR2 DEFAULT '']
    [, auto_commit IN BOOLEAN DEFAULT TRUE]);
```
RefreshTable Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>table_id</td>
<td>Name of the savepoint. The name is case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>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.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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 regular transaction affecting the table.

An exception is raised if the user does not have access to table_id, or 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
This 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 70–26 RefreshWorkspace Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE (the default) causes the operation to be executed as an autonomous</td>
</tr>
<tr>
<td></td>
<td>regular transaction that will be committed when it finishes.</td>
</tr>
<tr>
<td></td>
<td>FALSE causes the operation to be executed as part of the caller’s open</td>
</tr>
<tr>
<td></td>
<td>regular transaction (if one exists). If there is no open regular transaction,</td>
</tr>
<tr>
<td></td>
<td>the operation is executed in a new regular transaction. In either case, the</td>
</tr>
<tr>
<td></td>
<td>caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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.
RemoveWorkspace Procedure

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.

```sql
EXECUTE DBMS_WM.RefreshWorkspace ('NEWWORKSPACE');
```

RemoveWorkspace Procedure

This procedure rolls back the data in the workspace and removes all support structures created for the workspace. The workspace ceases to exist.

Syntax

```sql
DBMS_WM.RemoveWorkspace(
    workspace IN VARCHAR2,
    auto_commit IN BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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

This procedure removes the specified workspace and all its descendant workspaces. The data in the workspaces is rolled back and the workspace structure is removed.

#### Syntax

```
DBMS_WM.RemoveWorkspaceTree(
    workspace   IN VARCHAR2
[, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

#### 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.

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

This 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace to check for conflicts with other workspaces. The name is case sensitive.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to check for conflicts. The name is not case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>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.</td>
</tr>
</tbody>
</table>
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. `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

**Table 70–29 ResolveConflicts Procedure Parameters (Cont.)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep</td>
<td>Workspace in favor of which to resolve conflicts: PARENT, CHILD, or BASE.</td>
</tr>
<tr>
<td></td>
<td>PARENT causes the parent workspace rows to be copied to the child workspace.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
RevokeSystemPriv Procedure

LIVE workspace), and therefore the call to MergeWorkspace Procedure does not succeed.

5. ResolveConflicts 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. MergeWorkspace Procedure is called to merge Workspace1 changes to the LIVE workspace.

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 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

This 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]);
```
Summary of DBMS_WM Subprograms

Parameters

Contrast this procedure with 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 do not have the privilege to revoke `priv_types`.

Examples

The following example disallows user Smith from accessing workspaces and merging changes in workspaces.

```sql
EXECUTE DBMS_WM.RevokeSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE', 'Smith');
```
RevokeWorkspacePriv Procedure

This procedure revokes (removes) workspace-level privileges from users and roles for a specified workspace.

Syntax

```sql
DBMS_WM.RevokeWorkspacePriv(
    priv_types    IN VARCHAR2,
    workspace     IN VARCHAR2,
    grantee       IN VARCHAR2
[, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priv_types</td>
<td>A string of one or more keywords representing privileges. Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.</td>
</tr>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>grantee</td>
<td>Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td></td>
<td>TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes.</td>
</tr>
<tr>
<td></td>
<td>FALSE causes the operation to be executed as part of the caller’s open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

Usage Notes

Contrast this procedure with 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 do not have the privilege to revoke `priv_types`.

**Examples**

The following example disallows user `Smith` from accessing the `NEWWORKSPACE` workspace and merging changes in that workspace.

```sql
EXECUTE DBMS_WM.RevokeWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE', 'NEWWORKSPACE', 'Smith');
```

**RollbackResolve Procedure**

This procedure quits a conflict resolution session and discards all changes in the workspace since `BeginResolve Procedure` was executed.

**Syntax**

```sql
DBMS_WM.RollbackResolve(
    workspace IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure quits the current conflict resolution session (started by `BeginResolve Procedure`), and discards all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with `CommitResolve Procedure`, which saves all changes.

While the conflict resolution session is being rolled back, the workspace is frozen in `1WRITER` mode.

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 regular 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

This 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 70–33 RollbackTable Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>table_id</td>
<td>Name of the containing rows to be discarded. The name is not case sensitive.</td>
</tr>
<tr>
<td>sp_name</td>
<td>Name of the savepoint to which to roll back. The name is case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>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 all rows that meet the criteria of the other parameters are discarded.</td>
</tr>
</tbody>
</table>
```
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 regular transaction affecting **table_id** is active 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

This procedure discards all changes made after a specified savepoint in the workspace to all tables.
RollbackToSP Procedure

Syntax

```sql
DBMS_WM.RollbackToSP(
    workspace  IN VARCHAR2,
    savepoint_name  IN VARCHAR2
[, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>savepoint_name</td>
<td>Name of the savepoint to which to roll back changes. The name is case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
</tbody>
</table>

- **workspace**
  - **savepoint_name**
  - **auto_commit**

Usage Notes

While this procedure is executing, the workspace is frozen in NO_ACCESS mode.

Contrast this procedure with 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 active 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

This procedure discards all changes made in the workspace to all tables.

Syntax

DBMS_WM.RollbackWorkspace(
    workspace IN VARCHAR2
 [, auto_commit IN BOOLEAN DEFAULT TRUE]);

Parameters

Table 70–35 RollbackWorkspace Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>A boolean value (TRUE or FALSE).</td>
</tr>
<tr>
<td>commit</td>
<td>TRUE (the default) causes the operation to be executed as an autonomous</td>
</tr>
<tr>
<td></td>
<td>regular transaction that will be committed when it finishes.</td>
</tr>
<tr>
<td></td>
<td>FALSE causes the operation to be executed as part of the caller’s open</td>
</tr>
<tr>
<td></td>
<td>regular transaction (if one exists). If there is no open regular transaction</td>
</tr>
<tr>
<td></td>
<td>(if one exists). If there is no open regular transaction, the operation</td>
</tr>
<tr>
<td></td>
<td>is executed in a new regular transaction. In either case, the caller is</td>
</tr>
<tr>
<td></td>
<td>responsible for committing the transaction.</td>
</tr>
</tbody>
</table>

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 RollbackToSP Procedure, which rolls back changes to a specified savepoint.
Like `RemoveWorkspace Procedure`, RollbackWorkspace deletes the data in the workspace; however, unlike `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.

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 active in `workspace`.

**Examples**

The following example rolls back any changes made in the `NEWWORKSPACE` workspace since that workspace was created.

```sql
EXECUTE DBMS_WM.RollbackWorkspace ('NEWWORKSPACE');
```

**SetConflictWorkspace Procedure**

This procedure determines whether or not conflicts exist between a workspace and its parent.

**Syntax**

```sql
DBMS_WM.SetConflictWorkspace(
  workspace  IN VARCHAR2);
```

**Parameters**

**Table 70–36 SetConflictWorkspace Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure checks for any conflicts between `workspace` and its parent workspace, and it modifies the contents 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 may have been changed in the workspace, use the SQL statement `SELECT * FROM ALL_VERSIONED_TABLES`. 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.

```sql
EXECUTE DBMS_WM.SetConflictWorkspace ('B_focus_2');
```

**SetDiffVersions Procedure**

This 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**

```sql
DBMS_WM.SetDiffVersions(
    workspace1 IN VARCHAR2,
    workspace2 IN VARCHAR2);
```

or

```sql
DBMS_WM.SetDiffVersions(
    workspace1 IN VARCHAR2,
    savepoint1 IN VARCHAR2,
    workspace2 IN VARCHAR2,
    savepoint2 IN VARCHAR2);
```
Parameters

Table 70–37  SetDiffVersions Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace1</td>
<td>Name of the first workspace to be checked for differences in version-enabled tables. The name is case sensitive.</td>
</tr>
<tr>
<td>savepoint1</td>
<td>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.</td>
</tr>
<tr>
<td>workspace2</td>
<td>Name of the second workspace to be checked for differences in version-enabled tables. The name is case sensitive.</td>
</tr>
<tr>
<td>savepoint2</td>
<td>Name of the savepoint in workspace2 for which values are to be checked. The name is case sensitive.</td>
</tr>
</tbody>
</table>

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
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;
```
<table>
<thead>
<tr>
<th>MKT_ID</th>
<th>MKT_NAME</th>
<th>MANAGER</th>
<th>BUDGET</th>
<th>WM_DIFFVER</th>
<th>WM_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cola_a</td>
<td>Alvarez</td>
<td>2</td>
<td>DiffBase</td>
<td>NC</td>
</tr>
<tr>
<td>1</td>
<td>cola_a</td>
<td>Alvarez</td>
<td>1.5</td>
<td>B_focus_1, LATEST</td>
<td>U</td>
</tr>
<tr>
<td>1</td>
<td>cola_a</td>
<td>Alvarez</td>
<td>2</td>
<td>B_focus_2, LATEST</td>
<td>NC</td>
</tr>
<tr>
<td>2</td>
<td>cola_b</td>
<td>Burton</td>
<td>2</td>
<td>DiffBase</td>
<td>NC</td>
</tr>
<tr>
<td>2</td>
<td>cola_b</td>
<td>Beasley</td>
<td>3</td>
<td>B_focus_1, LATEST</td>
<td>U</td>
</tr>
<tr>
<td>2</td>
<td>cola_b</td>
<td>Burton</td>
<td>2.5</td>
<td>B_focus_2, LATEST</td>
<td>U</td>
</tr>
<tr>
<td>3</td>
<td>cola_c</td>
<td>Chen</td>
<td>1.5</td>
<td>DiffBase</td>
<td>NC</td>
</tr>
<tr>
<td>3</td>
<td>cola_c</td>
<td>Chen</td>
<td>1</td>
<td>B_focus_1, LATEST</td>
<td>U</td>
</tr>
<tr>
<td>3</td>
<td>cola_c</td>
<td>Chen</td>
<td>1.5</td>
<td>B_focus_2, LATEST</td>
<td>NC</td>
</tr>
<tr>
<td>4</td>
<td>cola_d</td>
<td>Davis</td>
<td>3.5</td>
<td>DiffBase</td>
<td>NC</td>
</tr>
<tr>
<td>4</td>
<td>cola_d</td>
<td>Davis</td>
<td>3</td>
<td>B_focus_1, LATEST</td>
<td>U</td>
</tr>
<tr>
<td>4</td>
<td>cola_d</td>
<td>Davis</td>
<td>2.5</td>
<td>B_focus_2, LATEST</td>
<td>U</td>
</tr>
</tbody>
</table>

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

This procedure enables access to versioned rows and to corresponding rows in the parent workspace.

Syntax

```sql
DBMS_WM.SetLockingOFF();
```

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.

```sql
EXECUTE DBMS_WM.SetLockingOFF;
```
SetLockingON Procedure

This procedure controls access to versioned rows and to corresponding rows in the previous version.

Syntax

DBMS_WM.SetLockingON(
   lockmode IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockmode</td>
<td>Locking mode. Must be E, S, or C.</td>
</tr>
</tbody>
</table>

- 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 `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.

```sql
EXECUTE DBMS_WM.SetLockingON ('E');
```

All rows locked by this user remain locked until the workspace is merged or rolled back.

**SetMultiWorkspaces Procedure**

This procedure makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.

**Syntax**

```sql
DBMS_WM.SetMultiWorkspaces(
    workspaces IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspaces</td>
<td>The workspace or workspaces for which information is to be added to the multiworkspace views. 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'</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure adds rows to the multiworkspace views (xxx_MS). 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.
- More than eight workspace names are specified in `workspaces`.

Examples

The following example adds information to the multiworkspace views for version-enabled tables in the `B_focus_1` workspace.

```sql
SQL> EXECUTE DBMS_WM.SetMultiWorkspaces ('B_focus_1');
```

SetWoOverwriteOFF Procedure

This 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

```sql
DBMS_WM.SetWoOverwriteOFF();
```

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 \texttt{<table_name>}_HIST views are described in \textit{Oracle9i Application Developer’s Guide - Workspace Manager}. The \texttt{VIEW_WO_OVERWRITE} and \texttt{VIEW_W_OVERWRITE} options are further described in the description of the \texttt{EnableVersioning Procedure}.

The history option affects the behavior of the \texttt{GotoDate Procedure}. See the Usage Notes for that procedure.

The result of the \texttt{SetWoOverwriteOFF} procedure remains in effect only for the duration of the current session. To reverse the effect of this procedure, use the \texttt{SetWoOverwriteON Procedure}.

### Examples

The following example disables the \texttt{VIEW_WO_OVERWRITE} history option.

\begin{verbatim}
EXECUTE DBMS_WM.SetWoOverwriteOFF;
\end{verbatim}

### SetWoOverwriteON Procedure

This procedure enables the \texttt{VIEW_WO_OVERWRITE} history option that had been disabled by the \texttt{SetWoOverwriteOFF Procedure}.

### Syntax

\begin{verbatim}
DBMS_WM.SetWoOverwriteON();
\end{verbatim}

### Usage Notes

This procedure affects the recording of history information in the views named \texttt{<table_name>}_HIST by changing the \texttt{VIEW_W_OVERWRITE} option to \texttt{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 \texttt{SetWoOverwriteOFF Procedure}.

The \texttt{<table_name>}_HIST views are described in \textit{Oracle9i Application Developer’s Guide - Workspace Manager}. The \texttt{VIEW_WO_OVERWRITE} and \texttt{VIEW_W_OVERWRITE} options are further described in the description of the \texttt{EnableVersioning Procedure}.

The \texttt{VIEW_W_OVERWRITE} history option can be overridden when a workspace is compressed by specifying the \texttt{compress_view_wo_overwrite} parameter as \texttt{TRUE} with the \texttt{CompressWorkspace Procedure} or \texttt{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

This procedure enables access to versioned rows in the specified workspace and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.SetWorkspaceLockModeOFF(
    workspace IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace for which to set the locking mode off. The name is case sensitive.</td>
</tr>
</tbody>
</table>

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 regular 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

This procedure sets the default mode for the row-level locking in the workspace.

Syntax

DBMS_WM.SetLockingON(
    workspace IN VARCHAR2,
    lockmode IN VARCHAR2,
    [, override IN BOOLEAN DEFAULT FALSE]);

Parameters

Table 70–41  SetWorkspaceLockModeON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace for which to set the locking mode. The name is case sensitive.</td>
</tr>
<tr>
<td>lockmode</td>
<td>Locking mode. Must be E, S, or C.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.)</td>
</tr>
<tr>
<td>override</td>
<td>A boolean value (TRUE or FALSE)</td>
</tr>
<tr>
<td></td>
<td>TRUE allows a session in the workspace to change the lockmode value by using the SetLockingON Procedure and SetLockingOFF Procedure.</td>
</tr>
<tr>
<td></td>
<td>FALSE (the default) prevents a session in the workspace from changing the lockmode value.</td>
</tr>
</tbody>
</table>
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.

There are no specific privileges associated with locking. Any session that can go to a workspace can set locking on.

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 regular transactions in workspace.
- workspace is a continually refreshed workspace (see the description of the isrefreshed parameter of the CreateWorkspace Procedure 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.

UnfreezeWorkspace Procedure

This procedure enables changes to a workspace, reversing the effect of FreezeWorkspace Procedure.
Summary of DBMS_WM Subprograms

Syntax

```
DBMS_WM.UnfreezeWorkspace(
    workspace  IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Name of the workspace. The name is case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

The operation fails if any sessions are active in `workspace`.

Only the owner of the workspace or a user with `WM_ADMIN_ROLE` can unfreeze a workspace. There are no specific privileges associated with freezing a workspace.

Examples

The following example unfreezes the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.UnfreezeWorkspace ('NEWWORKSPACE');
```

UnlockRows Procedure

This 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']);
```
UnlockRows Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>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.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table in which rows are to be unlocked. The name is not case sensitive.</td>
</tr>
<tr>
<td>where_clause</td>
<td>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.</td>
</tr>
<tr>
<td>all_or_user</td>
<td>Scope of the request: ALL or USER.</td>
</tr>
<tr>
<td></td>
<td>ALL: All locks accessible by the user in the current workspace are considered.</td>
</tr>
<tr>
<td></td>
<td>USER (default): Only locks owned by the user in the current workspace are considered.</td>
</tr>
<tr>
<td>lock_mode</td>
<td>Locking mode: E, S, or ES.</td>
</tr>
<tr>
<td></td>
<td>E: Only exclusive mode locks are considered.</td>
</tr>
<tr>
<td></td>
<td>S: Only shared mode locks are considered.</td>
</tr>
<tr>
<td></td>
<td>ES (default): Both exclusive mode and shared mode locks are considered.</td>
</tr>
</tbody>
</table>

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'');
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 Reference for more information on XML support and for an example of how to use DBMS_XMLGEN.

This chapter discusses the following topics:

- Summary of DBMS_XMLGEN Subprograms
### Summary of DBMS_XMLGEN Subprograms

#### Table 71–1 DBMS_XMLGEN Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;newContext Function&quot; on page 71-3</td>
<td>Creates a new context handle from a passed-in SQL query. The context handle can be used for the rest of the functions.</td>
</tr>
<tr>
<td>&quot;setRowTag Procedure&quot; on page 71-3</td>
<td>Sets the name of the element enclosing each row of the result. The default tag is ROW.</td>
</tr>
<tr>
<td>&quot;setRowSetTag Procedure&quot; on page 71-4</td>
<td>Sets the name of the element enclosing the entire result. The default tag is ROWSET.</td>
</tr>
<tr>
<td>&quot;getXML Procedure&quot; on page 71-5</td>
<td>Appends the XML to the CLOB passed in. Use the getNumRowsProcessed function to figure out if any rows were appended.</td>
</tr>
<tr>
<td>&quot;getXML Function&quot; on page 71-5</td>
<td>Returns the XML as a CLOB.</td>
</tr>
<tr>
<td>&quot;getNumRowsProcessed Function&quot; on page 71-6</td>
<td>Gets the number of SQL rows that were processed in the last call to getXML.</td>
</tr>
<tr>
<td>&quot;setMaxRows Procedure&quot; on page 71-7</td>
<td>Sets the maximum number of rows to be fetched each time.</td>
</tr>
<tr>
<td>&quot;setSkipRows Procedure&quot; on page 71-8</td>
<td>Sets the number of rows to skip every time before generating the XML. The default is 0.</td>
</tr>
<tr>
<td>&quot;setConvertSpecialChars Procedure&quot; on page 71-8</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;useItemTagsForColl Procedure&quot; on page 71-9</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;restartQUERY Procedure&quot; on page 71-10</td>
<td>Restarts the query to start fetching from the beginning.</td>
</tr>
<tr>
<td>&quot;closeContext Procedure&quot; on page 71-10</td>
<td>Closes the context and release all resources.</td>
</tr>
</tbody>
</table>
newContext Function

This function, given a query string, generates a new context handle to be used in subsequent functions.

Syntax

```sql
DBMS_XMLGEN.newContext (  
    queryString IN VARCHAR2)  
RETURN ctxHandle;
```

Parameters

Table 71–2 shows the parameters of the `newContext` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryString</td>
<td>The query string, the result of which must be converted to XML.</td>
</tr>
</tbody>
</table>

Returns

The context handle.

Usage Notes

You must call this function first to obtain a handle that you can use in the `getXML()` and other functions to get XML back from the result.

setRowTag Procedure

This procedure sets the name of the element separating all the rows. The default name is `ROW`.

Syntax

```sql
DBMS_XMLGEN.setRowTag (  
    ctx IN ctxHandle,  
    rowTag IN VARCHAR2);
```

Parameters

Table 71–3 shows the parameters of the `setRowTag` procedure.
setRowSetTag Procedure

Usage Notes

You can call this function to set the name of the root element of the document. The default name is ROWSET.

Syntax

```
DBMS_XMLGEN.setRowSetTag (
  ctx IN ctxHandle,
  rowSetTag IN VARCHAR2);
```

Parameters

Table 71–4 shows the parameters of the setRowSetTag procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle obtained from the newContext call.</td>
</tr>
<tr>
<td>rowSetTag (IN)</td>
<td>The name of the document element. NULL indicates that you do not want the ROW element present.</td>
</tr>
</tbody>
</table>

Usage Notes

You can call this function to set the name of the document root element if you do not want the default ROWSET name in the output. You can also set this to 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.
getXML Procedure

This procedure gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in.

Syntax

```sql
DBMS_XMLGEN.getXML (  
    ctx IN ctxHandle,  
    clobval IN OUT NCOPY clob,  
    dtdOrSchema IN number := NONE)  
RETURN boolean;
```

Parameters

Table 71–5 shows the parameters of the getXML procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle obtained from the newContext call.</td>
</tr>
<tr>
<td>clobval (IN/OUT)</td>
<td>The clob to which the XML document is appended.</td>
</tr>
<tr>
<td>dtdOrSchema (IN)</td>
<td>The Boolean to indicate generation of either a DTD or a schema.</td>
</tr>
</tbody>
</table>

Usage Notes

Use this version of the getXML function if you want to avoid any extra CLOB copies, and you want to reuse the same CLOB for subsequent calls. This getXML call is more efficient than the getXML function, although this involves creating the LOB locator.

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 function to check if any rows were retrieved.

getXML Function

This function generates the XML document and returns it as a CLOB.
getNumRowsProcessed Function

Syntax

```sql
DBMS_XMLGEN.getNumRowsProcessed (
   ctx IN ctxHandle
)
RETURN NUMBER;
```

Parameters

Table 71–7 shows the parameters of the getNumRowsProcessed function.

Table 71–7 getNumRowsProcessed Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle obtained from the newContext call.</td>
</tr>
<tr>
<td>dtdOrSchema (IN)</td>
<td>The Boolean to indicate generation of either a DTD or a schema.</td>
</tr>
</tbody>
</table>

Returns

A temporary CLOB containing the document.

Usage Notes

You must free the temporary CLOB obtained from this function using the `DBMS_LOB.FREETEMPORARY` call.
Table 71–7  getNumRowsProcessed Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryString</td>
<td>The query string, the result of which must be converted to XML.</td>
</tr>
</tbody>
</table>

Returns

The number of rows processed in the last call to getXML. This does not include the number of rows skipped.

Usage Notes

Use this function to determine the terminating condition if you are calling getXML in a loop. Note that getXML always generates an XML document, even if there are no rows present.

setMaxRows Procedure

This procedure sets the maximum number of rows to fetch from the SQL query result for every invocation of the getXML call.

Syntax

```sql
DBMS_XMLGEN.setMaxRows (  
    ctx IN ctxHandle,  
    maxRows IN NUMBER);  
```

Parameters

Table 71–8 shows the parameters of the setMaxRows procedure.

Table 71–8  SET_MAX_ROWS Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle corresponding to the query executed.</td>
</tr>
<tr>
<td>maxRows (IN)</td>
<td>The maximum number of rows to get per call to getXML.</td>
</tr>
</tbody>
</table>

Usage Notes

Closes all resources associated with this handle. After closing the context, you cannot use the handle for any other DBMS_XMLGEN function call.
setSkipRows Procedure

This procedure skips a given number of rows before generating the XML output for every call to the getXML routine.

Syntax

```
DBMS_XMLGEN.setSkipRows (  
    ctx IN ctxHandle,  
    skipRows IN NUMBER);
```

Parameters

Table 71–9 shows the parameters of the setSkipRows procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHandle (IN)</td>
<td>The context handle corresponding to the query executed.</td>
</tr>
<tr>
<td>skipRows (IN)</td>
<td>The number of rows to skip per call to getXML.</td>
</tr>
</tbody>
</table>

Usage Notes

You can use the skipRows parameter when generating paginated results for stateless Web pages using this utility. For example, when generating the first page of XML or HTML data, you can set skipRows to zero. For the next set, you can set the skipRows to the number of rows that you got in the first case.

setConvertSpecialChars Procedure

This procedure 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 &lt; . The default is to perform conversions.

Syntax

```
DBMS_XMLGEN.setConvertSpecialChars (  
    ctx IN ctxHandle,  
    conv IN boolean);
```

Parameters

Table 71–10 shows the parameters of the setConvertSpecialChars procedure.
Summary of DBMS_XMLGEN Subprograms

Table 71–10  setConvertSpecialChars Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle to use.</td>
</tr>
<tr>
<td>conv (IN)</td>
<td>True indicates that conversion is needed.</td>
</tr>
</tbody>
</table>

Usage Notes

You can use this function to speed up XML processing whenever you are sure that 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. In cases where the data is XML-safe, you can call this function to improve performance.

useItemTagsForColl Procedure

This procedure sets the 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.

Syntax

```
DBMS_XMLGEN.useItemTagsForColl (  
    ctx IN ctxHandle);
```

Parameters

Table 71–11 shows the parameters of the useItemTagsForColl procedure.

Table 71–11  useItemTagsForColl Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle.</td>
</tr>
</tbody>
</table>

Usage Notes

If you have a collection of NUMBER, the default tag name for the collection elements is NUMBER. Using this procedure, you can override this behavior and generate the collection column name with the _ITEM tag appended to it.
restartQUERY Procedure

This procedure restarts the query and generates the XML from the first row.

Syntax

```sql
DBMS_XMLGEN.restartQUERY ( ctx IN ctxHandle);
```

Parameters

Table 71–12 shows the parameters of the restartQuery procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle corresponding to the current query.</td>
</tr>
</tbody>
</table>

Usage Notes

You can call this procedure to start executing the query again, without having to create a new context.

closeContext Procedure

This procedure closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers.

Syntax

```sql
DBMS_XMLGEN.closeContext ( ctx IN ctxHandle);
```

Parameters

Table 71–13 shows the parameters of the closeContext procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx (IN)</td>
<td>The context handle to close.</td>
</tr>
</tbody>
</table>
Usage Notes

Closes all resources associated with this handle. After this you cannot use the handle for any other DBMS_XMLGEN function call.
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 Reference* for more information

This chapter discusses the following topics:

- Summary of DBMS_XMLQUERY Subprograms
## Summary of DBMS_XMLQUERY Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;newContext Function&quot; on page 72-3</td>
<td>Creates a query context and returns the context handle.</td>
</tr>
<tr>
<td>&quot;newContext Function&quot; on page 72-4</td>
<td>Creates a query context and returns the context handle.</td>
</tr>
<tr>
<td>&quot;closeContext Procedure&quot; on page 72-4</td>
<td>Closes or de-allocates a particular query context.</td>
</tr>
<tr>
<td>&quot;setRowsetTag Procedure&quot; on page 72-5</td>
<td>Sets the tag to be used to enclose the XML dataset.</td>
</tr>
<tr>
<td>&quot;setRowTag Procedure&quot; on page 72-5</td>
<td>Sets the tag to be used to enclose the XML element corresponding to a database record.</td>
</tr>
<tr>
<td>&quot;setErrorTag Procedure&quot; on page 72-6</td>
<td>Sets the tag to be used to enclose the XML error docs.</td>
</tr>
<tr>
<td>&quot;setRowIdAttrName Procedure&quot; on page 72-6</td>
<td>Sets the name of the id attribute of the row enclosing the tag.</td>
</tr>
<tr>
<td>&quot;setRowIdAttrValue Procedure&quot; on page 72-7</td>
<td>Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing the tag.</td>
</tr>
<tr>
<td>&quot;setCollIdAttrName Procedure&quot; on page 72-7</td>
<td>Sets the name of the id attribute of the separator tag of the collection element.</td>
</tr>
<tr>
<td>&quot;useNullAttributeIndicator Procedure&quot; on page 72-8</td>
<td>Specifies whether to use an XML attribute to indicate nullness, or to do it by omitting the particular entity in the XML document.</td>
</tr>
<tr>
<td>&quot;setTagCase Procedure&quot; on page 72-8</td>
<td>Specifies the case of the generated XML tags.</td>
</tr>
<tr>
<td>&quot;setDateFormat Procedure&quot; on page 72-9</td>
<td>Sets the format of the generated dates in the XML document.</td>
</tr>
<tr>
<td>&quot;setMaxRows Procedure&quot; on page 72-9</td>
<td>Sets the maximum number of rows to be converted to XML. By default, no maximum is set.</td>
</tr>
<tr>
<td>&quot;setSkipRows Procedure&quot; on page 72-10</td>
<td>Sets the number of rows to skip. By default, 0 rows are skipped.</td>
</tr>
<tr>
<td>&quot;setStylesheetHeader Procedure&quot; on page 72-10</td>
<td>Sets the stylesheet header (i.e. stylesheet processing instructions) in the generated XML document.</td>
</tr>
</tbody>
</table>
Table 72–1 DBMS_XMLQUERY Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;setXSLT Procedure&quot; on page 72-11</td>
<td>Registers a stylesheet to be applied to the generated XML.</td>
</tr>
<tr>
<td>&quot;setXSLT Procedure&quot; on page 72-12</td>
<td>Registers a stylesheet to be applied to the generated XML.</td>
</tr>
<tr>
<td>&quot;setBindValue Procedure&quot; on page 72-12</td>
<td>Sets a value for a particular bind name.</td>
</tr>
<tr>
<td>&quot;setMetaHeader Procedure&quot; on page 72-13</td>
<td>Sets the XML meta header.</td>
</tr>
<tr>
<td>&quot;setDataHeader Procedure&quot; on page 72-13</td>
<td>Sets the XML data header.</td>
</tr>
<tr>
<td>&quot;setRaiseException Procedure&quot; on page 72-14</td>
<td>Tells the XSU to throw the raised exceptions.</td>
</tr>
<tr>
<td>&quot;setNoRowsException Procedure&quot; on page 72-15</td>
<td>Tells the XSU whether or not to throw an OracleXMLNoRowsException when the XML document generated is empty.</td>
</tr>
<tr>
<td>&quot;propagateOriginalException Procedure&quot; on page 72-15</td>
<td>Tells the XSU that if an exception is raised, the XSU should throw that exception rather than wrapping it with an OracleXMLSQLException.</td>
</tr>
<tr>
<td>&quot;getExceptionContent Procedure&quot; on page 72-16</td>
<td>Returns the error code of the thrown exception and the error message (that is, the SQL error code).</td>
</tr>
<tr>
<td>&quot;getDTD Function&quot; on page 72-16</td>
<td>Generates the DTD based on the SQL query used to initialize the context.</td>
</tr>
<tr>
<td>&quot;getDTD Procedure&quot; on page 72-17</td>
<td>Generates the DTD based on the SQL query used to initialize the context.</td>
</tr>
<tr>
<td>&quot;getXML Function&quot; on page 72-17</td>
<td>Generates the XML document based on the SQL query used to initialize the context.</td>
</tr>
<tr>
<td>&quot;getXML Procedure&quot; on page 72-18</td>
<td>Generates the XML document based on the SQL query used to initialize the context.</td>
</tr>
</tbody>
</table>

**newContext Function**

This function creates a query context and returns the context handle.

**Syntax**

```
DBMS_XMLQUERY.newContext()
```
newContext Function

This function creates a query context and returns the context handle.

Syntax

```sql
DBMS_XMLQUERY.newContext (sqlQuery IN CLOB);
```

Returns

The context handle.

Parameters

Table 72–2 shows the parameters of the newContext function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlQuery (IN)</td>
<td>SQL query, the results of which to convert to XML</td>
</tr>
</tbody>
</table>

newContext Function

Table 72–2 newContext Function Parameters

closeContext Procedure

This procedure closes or de-allocates a particular query context.

Syntax

```sql
DBMS_XMLQUERY.closeContext (ctxHdl IN ctxType);
```
Parameters

Table 72–4 shows parameters of the `closeContext` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctxHdl</code> (IN)</td>
<td>The context handle.</td>
</tr>
</tbody>
</table>

**setRowsetTag Procedure**

This procedure sets the tag to be used to enclose the XML dataset.

**Syntax**

```sql
DBMS_XMLQUERY.setRowsetTag (    
    ctxHdl IN ctxType,    
    tag IN VARCHAR2);  
```

**Parameters**

Table 72–5 shows the parameters of the `setRowsetTag` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctxHdl</code> (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td><code>tag</code> (IN)</td>
<td>The tag name</td>
</tr>
</tbody>
</table>

**setRowTag Procedure**

This procedure sets the tag to be used to enclose the XML element corresponding to a database record.

**Syntax**

```sql
DBMS_XMLQUERY.setRowTag (    
    ctxHdl IN ctxType,    
    tag IN VARCHAR2);  
```

**Parameters**

Table 72–6 shows the parameters of the `setRowTag` procedure.
setErrorTag Procedure

This procedure sets the tag to be used to enclose the XML error docs.

Syntax

```sql
DBMS_XMLQUERY.setErrorTag (ctxHdl IN ctxType, tag IN VARCHAR2);
```

Parameters

Table 72–7 shows the parameters of the setErrorTag procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>tag (IN)</td>
<td>The tag name.</td>
</tr>
</tbody>
</table>

setRowIdAttrName Procedure

This procedure sets the name of the id attribute of the row enclosing the tag. Passing NULL or an empty string for the tag omits the row id attribute.

Syntax

```sql
DBMS_XMLQUERY.setRowIdAttrName (ctxHdl IN ctxType, attrName IN VARCHAR2);
```

Parameters

Table 72–8 shows the parameters of the setRowIdAttrName procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>attrName (IN)</td>
<td>The tag name.</td>
</tr>
</tbody>
</table>
setRowIdAttrValue Procedure

This procedure specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing the tag. Passing NULL or an empty string for the colName results in the row id attribute being assigned the row count value (that is, 0, 1, 2, and so on).

Syntax

```sql
DBMS_XMLQUERY.setRowIdAttrValue (
    ctxHdl IN ctxType,
    colName IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>colName (IN)</td>
<td>The column whose value is to be assigned to the row id attr.</td>
</tr>
</tbody>
</table>

setCollIdAttrName Procedure

This procedure sets the name of the id attribute of the separator tag of the collection element. Passing NULL or an empty string for the tag results omits the row id attribute.

Syntax

```sql
DBMS_XMLQUERY.setCollIdAttrName (
    ctxHdl IN ctxType,
    attrName IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>attrName (IN)</td>
<td>The attribute name.</td>
</tr>
</tbody>
</table>
useNullAttributeIndicator Procedure

Parameters

Table 72–10 shows the parameters of the setCollIdAttrName procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>attrName</td>
<td>The attribute name.</td>
</tr>
</tbody>
</table>

useNullAttributeIndicator Procedure

This procedure specifies whether to use an XML attribute to indicate nullness, or to do it by omitting the particular entity in the XML document.

Syntax

```sql
DBMS_XMLQUERY.useNullAttributeIndicator (ctxHdl IN ctxType, flag IN BOOLEAN);
```

Parameters

Table 72–11 shows the parameters of the useNullAttributeIndicator procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>Use attribute to indicate NULL?</td>
</tr>
</tbody>
</table>

setTagCase Procedure

This procedure specifies the case of the generated XML tags.

Syntax

```sql
DBMS_XMLQUERY.setTagCase (ctxHdl IN ctxType, tCase IN NUMBER);
```
Parameters

Table 72–12 shows the parameters of the setTagCase procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>tCase (IN)</td>
<td>The case of the tag (that is, 0-as Is, 1-lower, 2-upper).</td>
</tr>
</tbody>
</table>

**setDateFormat Procedure**

This procedure sets the format of the generated dates in the XML document. The syntax of the date format pattern (i.e. the date mask), should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to NULL or an empty string results in the use of the default mask, DEFAULT_DATE_FORMAT.

**Syntax**

```sql
DBMS_XMLQUERY.setDateFormat (
   ctxHdl  IN ctxType,
   mask    IN VARCHAR2);
```

Parameters

Table 72–13 shows the parameters of the setDateFormat procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>mask (IN)</td>
<td>The date mask.</td>
</tr>
</tbody>
</table>

**setMaxRows Procedure**

This procedure sets the maximum number of rows to be converted to XML. By default, no maximum is set.

**Syntax**

```sql
DBMS_XMLQUERY.setMaxRows (
   ctxHdl  IN ctxType,
);```
setSkipRows Procedure

```plsql
DBMS_XMLQUERY.setSkipRows (  
  ctxHdl IN ctxType,  
  rows IN NUMBER);
```

**Parameters**

Table 72–15 shows the parameters of the `setSkipRows` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>rows (IN)</td>
<td>The number of rows to skip.</td>
</tr>
</tbody>
</table>

### setSkipRows Procedure

This procedure sets the number of rows to skip. By default, 0 rows are skipped.

### Syntax

```plsql
DBMS_XMLQUERY.setSkipRows (  
  ctxHdl IN ctxType,  
  rows IN NUMBER);
```

**Parameters**

Table 72–15 shows the parameters of the `setSkipRows` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>rows (IN)</td>
<td>The number of rows to skip.</td>
</tr>
</tbody>
</table>

### setStylesheetHeader Procedure

This procedure sets the stylesheet header (that is, stylesheet processing instructions) in the generated XML document.
**Summary of DBMS_XMLQUERY Subprograms**

---

**Note:** Passing NULL for the uri argument will unset the stylesheet header and the stylesheet type.

### Syntax

```sql
DBMS_XMLQUERY.setStylesheetHeader (
    ctxHdl  IN ctxType,
    uri    IN VARCHAR2,
    type   IN VARCHAR2 := ‘text/xsl’);
```

### Parameters

Table 72–16 shows the parameters of the `setStylesheetHeader` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>uri</td>
<td>The stylesheet URL.</td>
</tr>
<tr>
<td>type</td>
<td>The stylesheet type, which defaults to text/xsl.</td>
</tr>
</tbody>
</table>

### setXSLT Procedure

This procedure registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it is replaced by the new one. To unregister the stylesheet, pass in a NULL for the uri argument.

```sql
DBMS_XMLQUERY.setXSLT (
    ctxHdl  IN ctxType,
    uri    IN VARCHAR2,
    ref    IN VARCHAR2 := null);
```

### Parameters

Table 72–17 shows the parameters of the `setXSLT` procedure.
setXSLT Procedure

This procedure registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it is replaced by the new one. To unregister the stylesheet, pass in a NULL or an empty string for the stylesheet argument.

Syntax

```sql
DBMS_XMLQUERY.setXSLT (
  ctxHdl IN ctxType,
  stylesheet CLOB,
  ref IN VARCHAR2 := null);
```

Parameters

Table 72–17 shows the parameters of the setXSLT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>uri (IN)</td>
<td>The stylesheet URL.</td>
</tr>
<tr>
<td>ref (IN)</td>
<td>The URL for include, import, and external entities.</td>
</tr>
</tbody>
</table>

setBindValue Procedure

This procedure sets a value for a particular bind name.

Syntax

```sql
DBMS_XMLQUERY.setBindValue (  
  ctxHdl IN ctxType,
  bindName IN VARCHAR2,
  bindValue IN VARCHAR2);  
```
Parameters

Table 72–19 shows the parameters of the setBindValue procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>bindName (IN)</td>
<td>The bind name.</td>
</tr>
<tr>
<td>bindValue (IN)</td>
<td>The bind value.</td>
</tr>
</tbody>
</table>

setMetaHeader Procedure

This procedure 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

DBMS_XMLQUERY.setMetaHeader (  
  ctxHdl IN ctxType,  
  header IN CLOB := null);  

Parameters

Table 72–20 shows the parameters of the setMetaHeader procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>header (IN)</td>
<td>The header.</td>
</tr>
</tbody>
</table>

setDataHeader Procedure

This procedure sets the XML data header. The data header is an XML entity that is appended at the beginning of the query-generated XML entity (that is, 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

```sql
DBMS_XMLQUERY.setDataHeader (  
    ctxHdl IN ctxType,  
    header IN CLOB := null,  
    tag IN VARCHAR2 := null);
```

Parameters

Table 72–21 shows the parameters of the `setDataHeader` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>header</td>
<td>The header.</td>
</tr>
<tr>
<td>tag</td>
<td>The tag used to enclose the data header and the rowset.</td>
</tr>
</tbody>
</table>

setRaiseException Procedure

This procedure tells the XSU to throw the raised exceptions. If this call is not made
or if false is passed to the flag argument, the XSU catches the SQL exceptions and
generates an XML document from the exception message.

Syntax

```sql
DBMS_XMLQUERY.setRaiseException (  
    ctxHdl IN ctxType,  
    flag IN BOOLEAN);
```

Parameters

Table 72–22 shows the parameters of the `setRaiseException` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>throw raised exceptions?</td>
</tr>
</tbody>
</table>
**setRaiseNoRowsException Procedure**

This procedure tells the XSU whether or not to throw an `OracleXMLNoRowsException` when the XML document generated is empty. By default, the exception is not thrown.

**Syntax**

```sql
DBMS_XMLQUERY.setRaiseNoRowsException (ctxHdl IN ctxType, flag IN BOOLEAN);
```

**Parameters**

Table 72–23 shows the parameters of the `setRaiseNoRowsException` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>flag (IN)</td>
<td>throw <code>OracleXMLNoRowsException</code> if no data?</td>
</tr>
</tbody>
</table>

**propagateOriginalException Procedure**

This procedure tells the XSU that if an exception is raised, the XSU should throw the exception raised rather than wrapping it with an `OracleXMLSQLException`.

**Syntax**

```sql
DBMS_XMLQUERY.propagateOriginalException (ctxHdl IN ctxType, flag IN BOOLEAN);
```

**Parameters**

Table 72–24 shows the parameters of the `propagateOriginalException` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>flag (IN)</td>
<td>Propagates the original exception?</td>
</tr>
</tbody>
</table>
getExceptionContent Procedure

This procedure, via its arguments, returns the error code of the thrown exception and the error message (that is, the 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

```sql
DBMS_XMLQUERY.getExceptionContent (  
   ctxHdl IN ctxType,  
   errNo OUT NUMBER,  
   errMsg OUT VARCHAR2);
```

Parameters

Table 72–25 shows the parameters of the getExceptionContent procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>errNo (OUT)</td>
<td>The error number.</td>
</tr>
<tr>
<td>errMsg (OUT)</td>
<td>The error message.</td>
</tr>
</tbody>
</table>

getDTD Function

This function generates the DTD based on the SQL query used to initialize the context.

Syntax

```sql
DBMS_XMLQUERY.getDTD (  
   ctxHdl IN ctxType,  
   withVer IN BOOLEAN := false);
```

Returns

The CLOB and the DTD.

Parameters

Table 72–26 shows the parameters of the getDTD function.
getDTD Procedure

This procedure generates the DTD based on the SQL query used to initialize the context.

Syntax

```
DBMS_XMLQUERY.getDTD (
  ctx  IN ctxType,
  xDoc  IN CLOB,
  withVer  IN BOOLEAN := false)
```

Parameters

Table 72–26 shows the parameters of the `getDTD` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>withVer (IN)</td>
<td>Generates the version information??</td>
</tr>
</tbody>
</table>

getXML Function

This function generates the XML document based on the SQL query used to initialize the context.

Syntax

```
DBMS_XMLQUERY.getXML (
  ctxHdl  IN ctxType,
  metaType  IN NUMBER := NONE)
```
getXML Procedure

Returns

The CLOB and the XML document.

Parameters

Table 72–28 shows the parameters of the getXML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>metaType (IN)</td>
<td>The XML metadata type (that is, none or DTD).</td>
</tr>
</tbody>
</table>

getXML Procedure

This procedure generates the XML document based on the SQL query used to initialize the context.

Syntax

```sql
DBMS_XMLQUERY.getXML (    
    ctxHdl IN ctxType, 
    xDoc IN CLOB, 
    metaType IN NUMBER := NONE) 
```

Parameters

Table 72–29 shows the parameters of the getXML procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc (IN)</td>
<td>The LOB in which to write the generated XML document.</td>
</tr>
<tr>
<td>metaType (IN)</td>
<td>The XML metadata type (I.e. none or DTD).</td>
</tr>
</tbody>
</table>
DBMS_XMLSAVE provides XML to database-type functionality.

This chapter discusses the following topics:

- Summary of DBMS_XMLSAVE Subprograms
Summary of DBMS/XMLSAVE Subprograms

### Table 73–1  DBMS/XMLSAVE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;newContext Function&quot; on page 73-3</td>
<td>Creates a save context and returns the context handle.</td>
</tr>
<tr>
<td>&quot;closeContext Procedure&quot; on page 73-3</td>
<td>Closes or de-allocates a particular save context.</td>
</tr>
<tr>
<td>&quot;setRowTag Procedure&quot; on page 73-4</td>
<td>Names the tag used in the XML document to enclose the XML elements corresponding to database records</td>
</tr>
<tr>
<td>&quot;setIgnoreCase Procedure&quot; on page 73-4</td>
<td></td>
</tr>
<tr>
<td>&quot;setDateFormat Procedure&quot; on page 73-5</td>
<td>Describes to the XSU the format of the dates in the XML document.</td>
</tr>
<tr>
<td>&quot;setBatchSize Procedure&quot; on page 73-5</td>
<td>Changes the batch size used during DML operations.</td>
</tr>
<tr>
<td>&quot;setCommitBatch Procedure&quot; on page 73-6</td>
<td>Sets the commit batch size.</td>
</tr>
<tr>
<td>&quot;setUpdateColumn Procedure&quot; on page 73-7</td>
<td>Adds a column to the update column list.</td>
</tr>
<tr>
<td>&quot;clearUpdateColumnList Procedure&quot; on page 73-7</td>
<td>Clears the update column list.</td>
</tr>
<tr>
<td>&quot;setKeyColumn Procedure&quot; on page 73-8</td>
<td>Adds a column to the key column list.</td>
</tr>
<tr>
<td>&quot;clearKeyColumnList Procedure&quot; on page 73-8</td>
<td>Clears the key column list.</td>
</tr>
<tr>
<td>&quot;insertXML Function&quot; on page 73-9</td>
<td>Inserts the XML document into the table specified at the context creation time.</td>
</tr>
<tr>
<td>&quot;insertXML Function&quot; on page 73-9</td>
<td>Inserts the XML document into the table specified at the context creation time.</td>
</tr>
<tr>
<td>&quot;updateXML Function&quot; on page 73-10</td>
<td>Updates the table specified at the context creation time with data from the XML document.</td>
</tr>
<tr>
<td>&quot;updateXML Function&quot; on page 73-10</td>
<td>Updates the table specified at the context creation time with data from the XML document.</td>
</tr>
</tbody>
</table>
Summary of DBMS.XMLSAVE Subprograms

Table 73–1  DBMS.XMLSAVE Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;deleteXML Function&quot; on page 73-11</td>
<td>Deletes records specified by data from the XML document, from the table specified at the context creation time.</td>
</tr>
<tr>
<td>&quot;deleteXML Function&quot; on page 73-11</td>
<td>Deletes records specified by data from the XML document, from the table specified at the context creation time.</td>
</tr>
</tbody>
</table>

newContext Function

This function creates a save context and returns the context handle.

Syntax

```sql
DBMS.XMLSAVE.newContext (   
    targetTable IN VARCHAR2);  
```

Parameters

Table 73–2 shows the parameters of the newContext function.

Table 73–2  newContext Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetTable</td>
<td>The target table into which the XML document is loaded.</td>
</tr>
</tbody>
</table>

Returns

The context handle.

closeContext Procedure

This procedure closes or de-allocates a save context.

Syntax

```sql
DBMS.XMLSAVE.closeContext (   
    ctxHdl IN ctxType);  
```

Parameters

Table 73–3 shows the parameters for the closeContext procedure.
setRowTag Procedure

This procedure names the tag used in the XML document to enclose the XML elements corresponding to database records.

Syntax

```
DBMS_XMLSAVE.setRowTag (  
    ctxHdl IN ctxType,  
    tag IN VARCHAR2);  
```

Parameters

*Table 73–4* shows the parameters of the `setRowTag` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctxHdl</code> (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td><code>tag</code> (IN)</td>
<td>The tag name.</td>
</tr>
</tbody>
</table>

setIgnoreCase Procedure

The XSU maps XML elements to database columns or attributes based on the element names (XML tags). The XSU mapping is case-insensitive.

Syntax

```
DBMS_XMLSAVE.setIgnoreCase (  
    ctxHdl IN ctxType,  
    flag IN NUMBER);  
```

Parameters

*Table 73–5* shows the parameters of the `setIgnoreCase` procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctxHdl</code> (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td><code>flag</code> (IN)</td>
<td>The flag value.</td>
</tr>
</tbody>
</table>
Summary of DBMS_XMLSAVE Subprograms

Table 73–5  setIgnoreCase Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>flag (IN)</td>
<td>Tag case is ignored: 1=true, 0=false.</td>
</tr>
</tbody>
</table>

setDateFormat Procedure

This procedure describes the format of the dates in the XML document to the XSU. The syntax of the date format pattern (the date mask) should conform to the requirements of the java.text.SimpleDateFormat class. If you set the mask to NULL or an empty string, the default mask, OracleXMLCore.DATE_FORMAT, is used.

Syntax

```sql
DBMS_XMLSAVE.setDateFormat (
    ctxHdl IN ctxType,
    mask IN VARCHAR2);
```

Parameters

Table 73–6 shows the parameters of the setDateFormat procedure.

Table 73–6  setDateFormat Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>mask (IN)</td>
<td>The date mask.</td>
</tr>
</tbody>
</table>

setBatchSize Procedure

This procedure changes the batch size used during DML operations. When performing inserts, updates, or deletes, batching the operations so that they are not executed as separate statements will result in performance gains. However, more memory is needed to buffer all the bind values. When batching is used, a commit occurs only after a batch is executed. Therefore, if one of the statements inside a batch fails, the whole batch is rolled back. If this behavior is unacceptable, set the batch size to 1.
setCommitBatch Procedure

Syntax

```sql
DBMS_XMLSAVE.setBatchSize (  
    ctxHdl     IN ctxType,  
    batchSize IN NUMBER);
```

Parameters

Table 73–7 shows the parameters of the `setBatchSize` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>batchSize (IN)</td>
<td>The batch size.</td>
</tr>
</tbody>
</table>

setCommitBatch Procedure

This procedure sets the commit batch size. The commit batch size refers to the number of records inserted after which a commit should follow. Note that if the `commitBatch` is less than 1 or the session is in autocommit mode, the XSU does not make any explicit commits. By default the commit-batch size is 0.

Syntax

```sql
DBMS_XMLSAVE.setCommitBatch (  
    ctxHdl     IN ctxType,  
    batchSize IN NUMBER);
```

Parameters

Table 73–8 shows the parameters of the `setCommitBatch` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>batchSize (IN)</td>
<td>The commit batch size.</td>
</tr>
</tbody>
</table>
setUpdateColumn Procedure

This procedure adds a column to the update column list. In the case of an insert, the default is to insert values in all the columns in the table. In case of updates, the default is to update only the columns corresponding to the tags present in the \textit{ROW} element of the XML document. When the update column list is specified, only the columns making up this list will be updated or inserted into.

Syntax

\begin{verbatim}
DBMS_XMLSAVE.setUpdateColumn ( 
  ctxHdl IN ctxType, 
  colName IN VARCHAR2);
\end{verbatim}

Parameters

\textit{Table 73–9} shows the parameters of the \textit{setUpdateColumn} procedure.

\textit{Table 73–9} \hspace{1em} \textit{setUpdateColumn} Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>colName (IN)</td>
<td>The column to be added to the update column list.</td>
</tr>
</tbody>
</table>

clearUpdateColumnList Procedure

This procedure clears the update column list.

Syntax

\begin{verbatim}
DBMS_XMLSAVE.clearUpdateColumnList ( 
  ctxHdl IN ctxType);
\end{verbatim}

Parameters

\textit{Table 73–10} shows the parameters of the \textit{clearUpdateColumnList} procedure.

\textit{Table 73–10} \hspace{1em} \textit{clearUpdateColumnList} Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
</tbody>
</table>
setKeyColumn Procedure

This procedure adds a column to the key column list. For updates or deletes, the columns in the key column list make up the WHERE clause of the update or delete statement. The key column list must be specified before updates can be done. The key column list is optional for delete operations.

Syntax

```
DBMS_XMLSAVE.setKeyColumn (  
  ctxHdl IN ctxType,  
  colName IN VARCHAR2);
```

Parameters

Table 73–11 shows the parameters of the setKeyColumn procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>colName (IN)</td>
<td>The column to be added to the key column list.</td>
</tr>
</tbody>
</table>

clearKeyColumnList Procedure

This procedure clears the key column list.

Syntax

```
DBMS_XMLSAVE.clearKeyColumnList (  
  ctxHdl IN ctxType);
```

Parameters

Table 73–12 shows the parameters of the clearKeyColumnList procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
</tbody>
</table>
Summary of DBMS_XMLSAVE Subprograms

insertXML Function

This function inserts the XML document into the table specified at the context creation time.

DBMS_XMLSAVE.insertXML (ctxHdl IN ctxType,
                           xDoc IN VARCHAR2);

Returns

The number of rows inserted.

Parameters

Table 73–13 shows the parameters of the insertXML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc</td>
<td>The string containing the XML document.</td>
</tr>
</tbody>
</table>

See Also: setKeyColumn Procedure on page 73–8.

insertXML Function

This function inserts the XML document into the table specified at the context creation time.

Syntax

DBMS_XMLSAVE.insertXML (ctxHdl IN ctxType,
                         xDoc IN CLOB);

Returns

The number of rows inserted.

Parameters

Table 73–14 shows the parameters of the insertXML function.
updateXML Function

This function updates the table specified at the context creation time with data from the XML document.

Syntax

```sql
DBMS_XMLSAVE.updateXML (
    ctxHdl IN ctxType,
    xDoc IN VARCHAR2);
```

Returns

The number of rows updated.

Parameters

Table 73–15 shows the parameters of the updateXML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc (IN)</td>
<td>The string containing the XML document.</td>
</tr>
</tbody>
</table>

updateXML Function

This function updates the table specified at the context creation time with data from the XML document.

Syntax

```sql
DBMS_XMLSAVE.updateXML (
    ctxHdl IN ctxType,
    xDoc IN CLOB);
```
Returns

The number of rows updated.

Parameters

Table 73–16 shows the parameters of the updateXML function.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc (IN)</td>
<td>The string containing the XML document.</td>
</tr>
</tbody>
</table>

deleteXML Function

This function deletes records specified by data from the XML document, from the table specified at the context creation time.

Syntax

```sql
DBMS_XMLSAVE.deleteXML (  
   ctxHdl IN ctxType,  
   xDoc IN VARCHAR2);  
```

Returns

The number of rows deleted.

Parameters

Table 73–17 shows the parameters of the deleteXML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc (IN)</td>
<td>The string containing the XML document.</td>
</tr>
</tbody>
</table>

deleteXML Function

This function deletes records specified by data from the XML document, from the table specified at the context creation time.
deleteXML Function

Syntax

```
DBMS_XMLSAVE.deleteXML (  
    ctxHdl IN ctxType,  
    xDoc IN CLOB);  
```

Returns

The number of rows deleted.

Parameters

Table 73–18 shows the parameters of the deleteXML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl (IN)</td>
<td>The context handle.</td>
</tr>
<tr>
<td>xDoc (IN)</td>
<td>The string containing the XML document.</td>
</tr>
</tbody>
</table>
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; e.g., 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 (e.g., 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;
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 75–1  IS_LOCATOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>Nested table or varray item.</td>
</tr>
</tbody>
</table>

Returns

Table 75–2  IS_LOCATOR Function Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collection item is indeed a locator.</td>
</tr>
<tr>
<td>0</td>
<td>Collection item is not a locator.</td>
</tr>
</tbody>
</table>

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 );
/

75-2  Supplied PL/SQL Packages and Types Reference
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;
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
Summary of UTL_ENCODE Subprograms

Table 76–1 UTL_ENCODE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BASE64_ENCODE Function&quot; on page 76-2</td>
<td>Encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string</td>
</tr>
<tr>
<td>&quot;BASE64_DECODE Function&quot; on page 76-3</td>
<td>Reads the base 64-encoded RAW input string and decodes it to its original RAW value</td>
</tr>
<tr>
<td>&quot;UUENCODE Function&quot; on page 76-3</td>
<td>Reads the RAW input string and encodes it to the corresponding uuencode format string</td>
</tr>
<tr>
<td>&quot;UUDECODE Function&quot; on page 76-4</td>
<td>Reads the RAW uuencode format input string and decodes it to the corresponding RAW string</td>
</tr>
<tr>
<td>&quot;QUOTED_PRINTABLE_ENCODE Function&quot; on page 76-5</td>
<td>Reads the RAW input string and encodes it to the corresponding quoted printable format string</td>
</tr>
<tr>
<td>&quot;QUOTED_PRINTABLE_DECODE Function&quot; on page 76-6</td>
<td>Reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string</td>
</tr>
</tbody>
</table>

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

```sql
FUNCTION base64_encode(
    r IN RAW
) RETURN RAW;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(base64_encode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

`r` is the RAW value to be encoded. There are no defaults or optional parameters.
Returns

Table 76–2  BASE64_ENCODE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the encoded base 64 elements</td>
</tr>
</tbody>
</table>

BASE64_DECODE Function

This function reads the base 64-encoded RAW input string and decodes it to its original RAW value.

Syntax

FUNCTION base64_decode (    
  r IN RAW)  
RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(base64_decode, WNDS, RNDS, WNPS, RNPS);

Parameters

r is the RAW string containing base 64-encoded data. There are no defaults or optional parameters.

Returns

Table 76–3  BASE64_DECODE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the decoded string</td>
</tr>
</tbody>
</table>

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 76-4.
UUDECODE Function

Syntax

```sql
FUNCTION uuencode(
    r IN RAW,
    type IN PLS_INTEGER DEFAULT 1,
    filename IN VARCHAR2 DEFAULT NULL,
    permission IN VARCHAR2 DEFAULT NULL) RETURN RAW;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(uuencode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>RAW string</td>
</tr>
<tr>
<td><code>type</code></td>
<td>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</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>Optional varchar2 parameter containing the uuencode filename; the default is <code>uuencode.txt</code></td>
</tr>
<tr>
<td><code>permission</code></td>
<td>Optional varchar2 parameter containing the permission mode; the default is 0 (a text string zero).</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the uuencode format string</td>
</tr>
</tbody>
</table>

UUDECODE Function

This function reads the `RAW` uuencode format input string and decodes it to the corresponding `RAW` string. See "UUENCODE Function" on page 76-3 for discussion of the cumulative nature of UUENCODE and UUDECODE for data streams.
Summary of UTL_ENCODE Subprograms

Syntax

FUNCTION uudecode(
    r IN RAW)
RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(uudecode, WNDS, RNDS, WNPS, RNPS);

Parameters

r is the RAW string containing the uuencoded data string. There are no defaults or optional parameters.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The decoded RAW string</td>
</tr>
</tbody>
</table>

QUOTED_PRINTABLE_ENCODE Function

This function reads the RAW input string and encodes it to the corresponding quoted printable format string.

Syntax

FUNCTION quoted_printable_encode(
    r IN RAW
RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(quoted_printable_encode, WNDS, RNDS, WNPS, RNPS);

Parameters

r is the RAW string. There are no defaults or optional parameters.
QUOTED_PRINTABLE_DECODE Function

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the quoted printable string</td>
</tr>
</tbody>
</table>

QUOTED_PRINTABLE_DECODE Function

This function reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string.

Syntax

```sql
FUNCTION quoted_printable_decode(
    r IN RAW
) RETURN RAW;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(quoted_printable_decode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

r is the RAW string containing a quoted printable data string. There are no defaults or optional parameters.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The decoded string</td>
</tr>
</tbody>
</table>
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
- File Ownership and Protections
- Types
- Exceptions
- Summary of UTL_FILE Subprograms
Security

The PL/SQL file I/O feature 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 can be accessed. Accessible directories must be specified in the instance parameter initialization file (INIT.ORA).

Accessible directories for the UTL_FILE functions are specified in the initialization file using the UTL_FILE_DIR parameter. For example:

UTL_FILE_DIR = <directory name>

---

**Note:** The directory specification is different on different platforms.

---

If the initialization file for the instance contains the line UTL_FILE_DIR = /usr/jsmith/my_app, then the directory /usr/jsmith/my_app is accessible to the FOPEN function. Note that a directory named /usr/jsmith/My_App would not be accessible on case-sensitive operating systems.

The parameter specification UTL_FILE_DIR = * should be used with caution. It turns off directory access checking and makes all directories accessible to the UTL_FILE functions.

---

**Caution:** Oracle does not recommend that you use the * option in production systems. Also, do not include ‘.’ (the current directory for UNIX) in the accessible directories list.

To ensure security on file systems that enable symbolic links, users must not be allowed WRITE permission to directories accessible by PL/SQL file I/O functions. The symbolic links and PL/SQL file I/O could be used to circumvent normal operating system permission checking and allow users read/write access to directories to which they would not otherwise have access.

---

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)
If the parameter initialization file contains only:

```sql
UTL_FILE_DIR=/appl/gl/log
UTL_FILE_DIR=/appl/gl/out
```

Then the following file locations and filenames are valid:

<table>
<thead>
<tr>
<th>FILE LOCATION</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>/appl/gl/log</td>
<td>L10324.log</td>
</tr>
<tr>
<td>/appl/gl/out</td>
<td>O10324.out</td>
</tr>
</tbody>
</table>

But the following file locations and filename are invalid:

<table>
<thead>
<tr>
<th>FILE LOCATION</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>/appl/gl/log/backup</td>
<td>L10324.log # subdirectory</td>
</tr>
<tr>
<td>/APPL/gl/log</td>
<td>L10324.log # uppercase</td>
</tr>
<tr>
<td>/appl/gl/log</td>
<td>backup/L10324.log # dir in name</td>
</tr>
<tr>
<td>/usr/tmp</td>
<td>T10324.tmp # not in INIT.ORA</td>
</tr>
</tbody>
</table>

---

**Caution:** There are no user-level file permissions. All file locations specified by the UTL_FILE_DIR parameters are valid for both reading and writing, for all users of the file I/O procedures. This can override operating system file permissions.

---

**Types**

```sql
TYPE file_type IS RECORD (id BINARY_INTEGER, datatype BINARY_INTEGER);
```

The contents of FILE_TYPE are private to the UTL_FILE package. You should not reference or change components of this record.
Exceptions

Procedures in UTL_FILE can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

Summary of UTL_FILE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;FOPEN Function&quot; on page 77-5</td>
<td>Opens a file for input or output with the default line size.</td>
</tr>
<tr>
<td>&quot;FOPEN Function&quot; on page 77-7</td>
<td>Opens a file with the maximum line size specified.</td>
</tr>
<tr>
<td>&quot;FOPEN_NCHAR Function&quot; on page 77-8</td>
<td>Opens a file in Unicode for input or output.</td>
</tr>
<tr>
<td>&quot;FOPEN_NCHAR Function&quot; on page 77-9</td>
<td>Opens a file in Unicode for input or output, with the maximum line size specified.</td>
</tr>
<tr>
<td>&quot;IS_OPEN Function&quot; on page 77-10</td>
<td>Determines if a file handle refers to an open file.</td>
</tr>
<tr>
<td>&quot;FCLOSE Procedure&quot; on page 77-11</td>
<td>Closes a file.</td>
</tr>
</tbody>
</table>

Table 77–1  UTL_FILE Package Exceptions

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_PATH</td>
<td>File location or filename was invalid.</td>
</tr>
<tr>
<td>INVALID_MODE</td>
<td>The open_mode parameter in FOPEN was invalid.</td>
</tr>
<tr>
<td>INVALID_FILEHANDLE</td>
<td>File handle was invalid.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File could not be opened or operated on as requested.</td>
</tr>
<tr>
<td>READ_ERROR</td>
<td>Operating system error occurred during the read operation.</td>
</tr>
<tr>
<td>WRITE_ERROR</td>
<td>Operating system error occurred during the write operation.</td>
</tr>
<tr>
<td>INTERNAL_ERROR</td>
<td>Unspecified PL/SQL error.</td>
</tr>
<tr>
<td>CHARSETMISMATCH</td>
<td>A file is opened using FOPEN_NCHAR, but later I/O operations use nonchar functions such as PUTF or GET_LINE.</td>
</tr>
</tbody>
</table>
Summary of UTL_FILE Subprograms

Table 77–2  UTL_FILE Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;FCLOSE_ALL Procedure&quot; on page 77-11</td>
<td>Closes all open file handles.</td>
</tr>
<tr>
<td>&quot;GET_LINE Procedure&quot; on page 77-12</td>
<td>Reads text from an open file.</td>
</tr>
<tr>
<td>&quot;GET_LINE_NCHAR Procedure&quot; on page 77-13</td>
<td>Reads text in Unicode from an open file.</td>
</tr>
<tr>
<td>&quot;PUT Procedure&quot; on page 77-13</td>
<td>Writes a string to a file.</td>
</tr>
<tr>
<td>&quot;PUT_NCHAR Procedure&quot; on page 77-14</td>
<td>Writes a Unicode string to a file.</td>
</tr>
<tr>
<td>&quot;NEW_LINE Procedure&quot; on page 77-15</td>
<td>Writes one or more operating system-specific line terminators to a file.</td>
</tr>
<tr>
<td>&quot;PUT_LINE Procedure&quot; on page 77-15</td>
<td>Writes a line to a file. This appends an operating system-specific line terminator.</td>
</tr>
<tr>
<td>&quot;PUT_LINE_NCHAR Procedure&quot; on page 77-16</td>
<td>Writes a Unicode line to a file.</td>
</tr>
<tr>
<td>&quot;PUTF Procedure&quot; on page 77-17</td>
<td>A PUT procedure with formatting.</td>
</tr>
<tr>
<td>&quot;PUTF_NCHARProcedure&quot; on page 77-18</td>
<td>A PUT_NCHAR procedure with formatting. Writes a Unicode string to a file, with formatting.</td>
</tr>
<tr>
<td>&quot;FFLUSH Procedure&quot; on page 77-19</td>
<td>Physically writes all pending output to a file.</td>
</tr>
</tbody>
</table>

**FOPEN Function**

This function opens a file for input or output. The file location must be an accessible directory, as defined in the instance’s initialization parameter UTL_FILE_DIR. The complete directory path must already exist; it is not created by FOPEN.

FOPEN returns a file handle, which must be used in all subsequent I/O operations on the file.

This version of FOPEN does not take a parameter for the maximum line size. Thus, the default (which is 1023 on most systems) is used. To specify a different maximum line size, use the other, overloaded version of “FOPEN Function” on page 77-7.
You can have a maximum of 50 files open simultaneously. See also "FOPEN_NCHAR Function" on page 77-8.

Syntax

```plsql
UTL_FILE.FOPEN (    location IN VARCHAR2,
    filename IN VARCHAR2,
    open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Operating system-specific string that specifies the directory in which to open the file.</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the file, including extension (file type), without any directory path information. (Under the UNIX operating system, the filename cannot be terminated with a <code>/</code>).</td>
</tr>
<tr>
<td>open_mode</td>
<td>String that specifies how the file should be opened (either upper or lower case letters can be used). The supported values, and the UTL_FILE procedures that can be used with them are: r read text (GET_LINE) w write text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH) a append text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)</td>
</tr>
</tbody>
</table>

Note: If you open a file that does not exist using the a value for open_mode, then the file is created in write (w) mode.

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.
Summary of UTL_FILE Subprograms

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 in the initialization file. 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 in the initialization file.

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

Exceptions

INVALID_PATH
INVALID_MODE
INVALID_OPERATION

FOPEN Function

This function opens a file. With this version of FOPEN, you can specify the maximum line size. The other version of the "FOPEN Function" on page 77-5 uses the default line size. You can have a maximum of 50 files open simultaneously.

See also "FOPEN_NCHAR Function" on page 77-9.

Syntax

UTL_FILE.FOPEN (  
    location   IN VARCHAR2,  
    filename   IN VARCHAR2,  
    open_mode  IN VARCHAR2,  
    max_linesize IN BINARY_INTEGER)  
RETURN file_type;

Parameters

Table 77–4  FOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of file.</td>
</tr>
</tbody>
</table>
FOPEN_NCHAR Function

Table 77–4  FOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>File name (including extension).</td>
</tr>
<tr>
<td>open_mode</td>
<td>Open mode (r, w, a).</td>
</tr>
<tr>
<td>max_linesize</td>
<td>Maximum number of characters per line, including the newline character, for this file. (minimum value 1, maximum value 32767).</td>
</tr>
</tbody>
</table>

Returns

Table 77–5  FOPEN Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_type</td>
<td>Handle to open file.</td>
</tr>
</tbody>
</table>

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 this function, you can read or write a text file in Unicode instead of in the database charset. See also "FOPEN Function" on page 77-5.

Syntax

```
UTL_FILE.FOPEN_NCHAR (   location  IN VARCHAR2,   filename  IN VARCHAR2,   open_mode  IN VARCHAR2)   RETURN UTL_FILE.FILE_TYPE;
```
Table 77–6  FOPEN_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Operating system-specific string that specifies the directory in which to open the file.</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the file, including extension (file type), without any directory path information. (Under the UNIX operating system, the filename cannot be terminated with a /).</td>
</tr>
<tr>
<td>open_mode</td>
<td>String that specifies how the file should be opened (either upper or lower case letters can be used). The supported values, and the UTL_FILE procedures that can be used with them are: r read text (GET_LINE_NCHAR) w write text (PUT_NCHAR, PUT_LINE_NCHAR, NEW_LINE, PUTF_NCHAR, FFLUSH) a append text (PUT_NCHAR, PUT_LINE_NCHAR, NEW_LINE, PUTF_NCHAR, FFLUSH)</td>
</tr>
</tbody>
</table>

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 77-7.

Syntax

```sql
UTL_FILE.FOPEN_NCHAR (  
    location IN VARCHAR2,  
    filename IN VARCHAR2,  
    open_mode IN VARCHAR2,  
    max_linesize IN BINARY_INTEGER)  
RETURN file_type;
```
Parameters

Table 77–7  FOPEN_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of file.</td>
</tr>
<tr>
<td>filename</td>
<td>File name (including extension).</td>
</tr>
<tr>
<td>open_mode</td>
<td>Open mode (r, w, a).</td>
</tr>
<tr>
<td>max_linesize</td>
<td>Maximum number of characters per line, including the newline character, for this file. (minimum value 1, maximum value 32767).</td>
</tr>
</tbody>
</table>

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

```sql
UTL_FILE.IS_OPEN ( file IN FILE_TYPE)
RETURN BOOLEAN;
```

Parameters

Table 77–8  IS_OPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN or FOPEN_NCHAR call.</td>
</tr>
</tbody>
</table>

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 77–9  FCLOSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN or FOPEN_NCHAR call.</td>
</tr>
</tbody>
</table>

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.
GET_LINE Procedure

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.

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 an input record is 1023 bytes, unless you specify a larger size in the overloaded version of FOPEN. See also "GET_LINE_NCHAR Procedure" on page 77-13.

Syntax

```sql
UTL_FILE.GET_LINE (  
    file    IN  FILE_TYPE,  
    buffer  OUT VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN call. The file must be open for reading (mode r), otherwise an INVALID_OPERATION exception is raised.</td>
</tr>
<tr>
<td>buffer</td>
<td>Data buffer to receive the line read from the file.</td>
</tr>
</tbody>
</table>

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 77-12.

Syntax

```sql
UTL_FILE.GET_LINE_NCHAR ( 
    file IN FILE_TYPE, 
    buffer OUT NVARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>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.</td>
</tr>
<tr>
<td>buffer</td>
<td>Data buffer to receive the line read from the file.</td>
</tr>
</tbody>
</table>

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.

The maximum size of an input record is 1023 bytes, unless you specify a larger size in the overloaded version of FOPEN. See also "PUT_NCHAR Procedure" on page 77-14.

Syntax

```sql
UTL_FILE.PUT ( 
    file IN FILE_TYPE, 
    buffer IN VARCHAR2);
```
PUT_NCHAR Procedure

Parameters

*Table 77–12  PUT Procedure Parameters*

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>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.</td>
</tr>
<tr>
<td>buffer</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 77-13.

Syntax

```sql
UTL_FILE.PUT_INCHAR (
    file IN FILE_TYPE,
    buffer IN NVARCHAR2);
```

Parameters

*Table 77–13  PUT_NCHAR Procedure Parameters*

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>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.</td>
</tr>
<tr>
<td>buffer</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN or FOPEN_NCHAR call.</td>
</tr>
<tr>
<td>lines</td>
<td>Number of line terminators to be written to the file.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_FILEHANDLE</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
</tr>
<tr>
<td>WRITE_ERROR</td>
</tr>
</tbody>
</table>

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 for an output record is 1023 bytes, unless you specify a larger value using the overloaded version of FOPEN. See also "PUT_LINE_NCHAR Procedure" on page 77-16.

Syntax

```
UTL_FILE.PUT_LINE (
    file    IN FILE_TYPE,
    buffer  IN VARCHAR2);
```
PUT_LINE_NCHAR Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN call.</td>
</tr>
<tr>
<td>buffer</td>
<td>Text buffer that contains the lines to be written to the file.</td>
</tr>
</tbody>
</table>

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 77-15.

Syntax

```sql
UTL_FILE.PUT_LINE_NCHAR (  
    file    IN FILE_TYPE,  
    buffer  IN NVARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>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.</td>
</tr>
<tr>
<td>buffer</td>
<td>Text buffer that contains the lines to be written to the file.</td>
</tr>
</tbody>
</table>
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.

%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 77-18.

Syntax

UTL_FILE.PUTF (  
    file IN FILE_TYPE,  
    format IN VARCHAR2,  
    [arg1 IN VARCHAR2 DEFAULT NULL,  
    ...  
    arg5 IN VARCHAR2 DEFAULT NULL]);

Parameters

Table 77–17  PUTF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN call.</td>
</tr>
<tr>
<td>format</td>
<td>Format string that can contain text as well as the formatting characters \n and %s.</td>
</tr>
<tr>
<td>arg1..arg5</td>
<td>From one to five operational argument strings.</td>
</tr>
</tbody>
</table>

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 77-17.

Syntax

UTL_FILE.PUTF_NCHAR (  
   file     IN FILE_TYPE,
   format   IN NVARCHAR2,
   [arg1    IN NVARCHAR2 DEFAULT NULL,
   ...      
   arg5    IN NVARCHAR2 DEFAULT NULL];

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>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.</td>
</tr>
<tr>
<td>format</td>
<td>Format string that can contain text as well as the formatting characters \n and %s.</td>
</tr>
</tbody>
</table>
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 77–19  FFLUSH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN or FOPEN_NCHAR call.</td>
</tr>
</tbody>
</table>
```

Exceptions

- INVALID_FILEHANDLE
- INVALID_OPERATION
- WRITE_ERROR
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.

The package contains a set of APIs that enables users to write PL/SQL programs that communicate with Web (HTTP) servers. **UTL_HTTP** also contains a function that can be used in SQL queries. Besides HTTP, it also supports HTTP over the Secured Socket Layer protocol (SSL), also known as HTTPS, directly or via 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 84, "UTL_URL"
- Chapter 82, "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 Exceptions**
- **UTL_HTTP Examples**
- **Summary of UTL_HTTP Subprograms**
UTL_HTTP Constants, Types and Flow

UTL_HTTP Constants

Table 78–1 lists the defined constants for UTL_HTTP.

<table>
<thead>
<tr>
<th>Constant and Syntax</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VERSION_1_0 CONSTANT VARCHAR2(10) := 'HTTP/1.0';</td>
<td>Denotes HTTP version 1.0 that can be used in the function begin_request.</td>
</tr>
<tr>
<td>HTTP_VERSION_1 CONSTANT VARCHAR2(10) := 'HTTP/1.1';</td>
<td>Denotes HTTP version 1.1 that can be used in the function begin_request.</td>
</tr>
<tr>
<td>DEFAULT_HTTP_PORT CONSTANT PLS_INTEGER := 80;</td>
<td>The default TCP/IP port (80) at which a Web server or proxy server listens</td>
</tr>
<tr>
<td>DEFAULT_HTTPS_PORT CONSTANT PLS_INTEGER := 443;</td>
<td>The default TCP/IP port (443) at which an HTTPS Web server listens</td>
</tr>
</tbody>
</table>

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;
HTTPSEE_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 78–1 UTL_HTTP Constants

<table>
<thead>
<tr>
<th>Constant and Syntax</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_BAD_REQUEST CONSTANT PLS_INTEGER := 400;</td>
<td></td>
</tr>
<tr>
<td>HTTP_UNAUTHORIZED CONSTANT PLS_INTEGER := 401;</td>
<td></td>
</tr>
<tr>
<td>HTTP_PAYMENT_REQUIRED CONSTANT PLS_INTEGER := 402;</td>
<td></td>
</tr>
<tr>
<td>HTTP_FORBIDDEN CONSTANT PLS_INTEGER := 403;</td>
<td></td>
</tr>
<tr>
<td>HTTP_NOT_FOUND CONSTANT PLS_INTEGER := 404;</td>
<td></td>
</tr>
<tr>
<td>HTTP_NOT_ACCEPTABLE CONSTANT PLS_INTEGER := 406;</td>
<td></td>
</tr>
<tr>
<td>HTTP_PROXY_AUTH_REQUIRED CONSTANT PLS_INTEGER := 407;</td>
<td></td>
</tr>
<tr>
<td>HTTP_REQUEST_TIME_OUT CONSTANT PLS_INTEGER := 408;</td>
<td></td>
</tr>
<tr>
<td>HTTP_CONFLICT CONSTANT PLS_INTEGER := 409;</td>
<td></td>
</tr>
<tr>
<td>HTTP_GONE CONSTANT PLS_INTEGER := 410;</td>
<td></td>
</tr>
<tr>
<td>HTTP_LENGTH_REQUIRED CONSTANT PLS_INTEGER := 411;</td>
<td></td>
</tr>
<tr>
<td>HTTP_PRECONDITION_FAILED CONSTANT PLS_INTEGER := 412;</td>
<td></td>
</tr>
<tr>
<td>HTTP_REQUEST_ENTITY_TOO_LARGE CONSTANT PLS_INTEGER := 413;</td>
<td></td>
</tr>
<tr>
<td>HTTP_REQUEST_URI_TOO_LARGE CONSTANT PLS_INTEGER := 414;</td>
<td></td>
</tr>
<tr>
<td>HTTP_UNSUPPORTED_MEDIA_TYPE CONSTANT PLS_INTEGER := 415;</td>
<td></td>
</tr>
<tr>
<td>HTTP_REQ_RANGE_NOT_SATISFIABLE CONSTANT PLS_INTEGER := 416;</td>
<td></td>
</tr>
<tr>
<td>HTTP_EXPECTATION_FAILED CONSTANT PLS_INTEGER := 417;</td>
<td></td>
</tr>
<tr>
<td>HTTP_NOT_IMPLEMENTED CONSTANT PLS_INTEGER := 501;</td>
<td></td>
</tr>
<tr>
<td>HTTP_BAD_GATEWAY CONSTANT PLS_INTEGER := 502;</td>
<td></td>
</tr>
<tr>
<td>HTTP_SERVICE_UNAVAILABLE CONSTANT PLS_INTEGER := 503;</td>
<td></td>
</tr>
<tr>
<td>HTTP_GATEWAY_TIME_OUT CONSTANT PLS_INTEGER := 504;</td>
<td></td>
</tr>
<tr>
<td>HTTP_VERSION_NOT_SUPPORTED CONSTANT PLS_INTEGER := 505;</td>
<td></td>
</tr>
</tbody>
</table>

UTL_HTTP Types

Use the following types with UTL_HTTP.

REQ Type

Use this PL/SQL record type to represent an HTTP request.
### Syntax

```sql
TYPE req IS RECORD (  
    url VARCHAR2(32767),  
    method VARCHAR2(64),  
    http_version VARCHAR2(64),  
);
```

### Parameters

**Table 78–2** shows the parameters for the REQ type.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the HTTP request. It is set after the request is created by <code>begin_request</code>.</td>
</tr>
<tr>
<td>method</td>
<td>The method to be performed on the resource identified by the URL. It is set after the request is created by <code>begin_request</code>.</td>
</tr>
<tr>
<td>http_version</td>
<td>The HTTP protocol version used to send the request. It is set after the request is created by <code>begin_request</code>.</td>
</tr>
</tbody>
</table>

### 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

```sql
TYPE resp IS RECORD (  
    status_code PLS_INTEGER,  
    reason_phrase VARCHAR2(256),  
    http_version VARCHAR2(64),  
);
```

#### Parameters

**Table 78–3** shows the parameters for the RESP type.
**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**

```sql
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 78–4` shows the fields for the `COOKIE` and `COOKIE_TABLE` record types.
Table 78–4  Fields of COOKIE and COOKIE_TABLE Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the HTTP cookie</td>
</tr>
<tr>
<td>value</td>
<td>The value of the cookie</td>
</tr>
<tr>
<td>domain</td>
<td>The domain for which the cookie is valid</td>
</tr>
<tr>
<td>expire</td>
<td>The time by which the cookie will expire</td>
</tr>
<tr>
<td>path</td>
<td>The subset of URLs to which the cookie applies</td>
</tr>
<tr>
<td>secure</td>
<td>Should the cookie be returned to the Web server using secured means only.</td>
</tr>
<tr>
<td>version</td>
<td>The version of the HTTP cookie specification the cookie conforms. This field is NULL for Netscape cookies.</td>
</tr>
<tr>
<td>comment</td>
<td>The comment that describes the intended use of the cookie. This field is NULL for Netscape cookies.</td>
</tr>
</tbody>
</table>

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 78-14.

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**

```plsql
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, or an exception will be raised.
Figure 78–1 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`
Table 78–5 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 78–5  UTL_HTTP Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
<th>Where Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_failed</td>
<td>29273</td>
<td>The request fails to executes</td>
<td>Any HTTP request or response API when detailed_exception is disabled</td>
</tr>
<tr>
<td>bad_argument</td>
<td>29261</td>
<td>The argument passed to the API is bad</td>
<td>Any HTTP request or response API when detailed_exception is enabled</td>
</tr>
<tr>
<td>bad_url</td>
<td>29262</td>
<td>The requested URL is badly formed</td>
<td>begin_request, when detailed_exception is enabled</td>
</tr>
<tr>
<td>protocol_error</td>
<td>29263</td>
<td>An HTTP protocol error occurs when communicating with the Web server</td>
<td>set_header, get_response, read_raw, read_text, and read_line, when detailed_exception is enabled</td>
</tr>
<tr>
<td>unknown_scheme</td>
<td>29264</td>
<td>The scheme of the requested URL is unknown</td>
<td>begin_request and get_response, when detailed_exception is enabled</td>
</tr>
<tr>
<td>header_not_found</td>
<td>29265</td>
<td>The header is not found</td>
<td>get_header, get_header_by_name, when detailed_exception is enabled</td>
</tr>
<tr>
<td>end_of_body</td>
<td>29266</td>
<td>The end of HTTP response body is reached</td>
<td>read_raw, read_text, and read_line, when detailed_exception is enabled</td>
</tr>
</tbody>
</table>
### Table 78-5 UTL_HTTP Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
<th>Where Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>illegal_call</td>
<td>29267</td>
<td>The call to UTL_HTTP is illegal at the current state of the HTTP request</td>
<td>set_header, set_authentication, and set_persistent_conn_support, when detailed_exception is enabled</td>
</tr>
<tr>
<td>http_client_error</td>
<td>29268</td>
<td>The response status code indicates that a client error has occurred</td>
<td>get_response, when detailed_exception is enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(status code in 4xx range)</td>
<td></td>
</tr>
<tr>
<td>http_server_error</td>
<td>29269</td>
<td>The response status code indicates that a client error has occurred</td>
<td>get_response, when detailed_exception is enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(status code in 5xx range)</td>
<td></td>
</tr>
<tr>
<td>too_many_requests</td>
<td>29270</td>
<td>Too many requests or responses are open</td>
<td>begin_request, when detailed_exception is enabled</td>
</tr>
<tr>
<td>partial_multibyte exception</td>
<td>29275</td>
<td>No complete character is read and a partial multi-byte character is found at the end of the response body</td>
<td>read_text and read_line, when detailed_exception is enabled</td>
</tr>
<tr>
<td>transfer_timeout</td>
<td>29276</td>
<td>No data is read and a read timeout occurred</td>
<td>read_text and read_line, when detailed_exception is enabled</td>
</tr>
</tbody>
</table>

**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);
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

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;

    END LOOP;
    utl_http.set_cookies(my_session_id, secure);

END;
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;
### Table 78–6  UTL_HTTP Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple HTTP fetches in a single call</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;REQUEST Function&quot; on page 78-20</td>
<td>Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.</td>
</tr>
<tr>
<td>&quot;REQUEST_PIECES Function&quot; on page 78-22</td>
<td>Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.</td>
</tr>
<tr>
<td><strong>Session Settings</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;SET_PROXY Procedure&quot; on page 78-26</td>
<td>Sets the proxy to be used for requests of HTTP or other protocols</td>
</tr>
<tr>
<td>&quot;GET_PROXY Procedure&quot; on page 78-27</td>
<td>Retrieves the current proxy settings</td>
</tr>
<tr>
<td>&quot;SET_COOKIE_SUPPORT Procedure&quot; on page 78-27</td>
<td>Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session</td>
</tr>
<tr>
<td>&quot;GET_COOKIE_SUPPORT Procedure&quot; on page 78-28</td>
<td>Retrieves the current cookie support settings</td>
</tr>
<tr>
<td>&quot;SET_FOLLOW_REDIRECT Procedure&quot; on page 78-29</td>
<td>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</td>
</tr>
<tr>
<td>&quot;GET_FOLLOW_REDIRECT Procedure&quot; on page 78-30</td>
<td>Retrieves the follow-redirect setting in the current session</td>
</tr>
<tr>
<td>&quot;SET_BODY_CHARSET Procedure&quot; on page 78-30</td>
<td>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</td>
</tr>
<tr>
<td>&quot;GET_BODY_CHARSET Procedure&quot; on page 78-31</td>
<td>Retrieves the default character set of the body of all future HTTP requests</td>
</tr>
<tr>
<td>&quot;SET_PERSISTENT_CONN_SUPPORT Procedure&quot; on page 78-32</td>
<td>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</td>
</tr>
<tr>
<td>&quot;GET_PERSISTENT_CONN_SUPPORT Procedure&quot; on page 78-34</td>
<td>Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session</td>
</tr>
<tr>
<td>&quot;SET_RESPONSE_ERROR_CHECK Procedure&quot; on page 78-35</td>
<td>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</td>
</tr>
</tbody>
</table>
Table 78–6  UTL_HTTP Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GET_RESPONSE_ERROR_CHECK Procedure&quot; on page 78-36</td>
<td>Checks if the response error check is set or not</td>
</tr>
<tr>
<td>&quot;SET_DETAILED_EXCP_SUPPORT Procedure&quot; on page 78-36</td>
<td>Sets the UTL_HTTP package to raise a detailed exception</td>
</tr>
<tr>
<td>&quot;GET_DETAILED_EXCP_SUPPORT Procedure&quot; on page 78-37</td>
<td>Checks if the UTL_HTTP package will raise a detailed exception or not</td>
</tr>
<tr>
<td>&quot;SET_WALLET Procedure&quot; on page 78-37</td>
<td>Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS</td>
</tr>
<tr>
<td>&quot;SET_TRANSFER_TIMEOUT Procedure&quot; on page 78-38</td>
<td>Sets the timeout value for UTL_HTTP to read the HTTP response from the Web server or proxy server</td>
</tr>
<tr>
<td>&quot;GET_TRANSFER_TIMEOUT Procedure&quot; on page 78-39</td>
<td>Retrieves the current network transfer timeout value</td>
</tr>
</tbody>
</table>

HTTP Requests

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BEGIN_REQUEST Function&quot; on page 78-39</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;SET_HEADER Procedure&quot; on page 78-40</td>
<td>Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.</td>
</tr>
<tr>
<td>&quot;SET_AUTHENTICATION Procedure&quot; on page 78-41</td>
<td>Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.</td>
</tr>
<tr>
<td>&quot;SET_COOKIE_SUPPORT Procedure&quot; on page 78-42</td>
<td>Enables or disables support for the HTTP cookies in the request.</td>
</tr>
<tr>
<td>&quot;SET_FOLLOW_REDIRECT Procedure&quot; on page 78-43</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;SET_BODY_CHARSET Procedure&quot; on page 78-44</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 78–6  UTL_HTTP Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET_PERSISTENT_CONN_SUPPORT Procedure&quot; on page 78-45</td>
<td>Enables or disables support for the HTTP 1.1 persistent-connection in the request.</td>
</tr>
<tr>
<td>&quot;WRITE_TEXT Procedure&quot; on page 78-47</td>
<td>Writes some text data in the HTTP request body.</td>
</tr>
<tr>
<td>&quot;WRITE_LINE Procedure&quot; on page 78-48</td>
<td>Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP).</td>
</tr>
<tr>
<td>&quot;WRITE_RAW Procedure&quot; on page 78-50</td>
<td>Writes some binary data in the HTTP request body.</td>
</tr>
<tr>
<td>&quot;END_REQUEST Procedure&quot; on page 78-51</td>
<td>Ends the HTTP request.</td>
</tr>
</tbody>
</table>

HTTP Responses

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GET_RESPONSE Function&quot; on page 78-51</td>
<td>Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed.</td>
</tr>
<tr>
<td>&quot;GET_HEADER_COUNT Function&quot; on page 78-52</td>
<td>Returns the number of HTTP response headers returned in the response.</td>
</tr>
<tr>
<td>&quot;GET_HEADER Procedure&quot; on page 78-52</td>
<td>Returns the n\textsuperscript{th} HTTP response header name and value returned in the response.</td>
</tr>
<tr>
<td>&quot;GET_HEADER_BY_NAME Procedure&quot; on page 78-53</td>
<td>Returns the HTTP response header value returned in the response given the name of the header.</td>
</tr>
<tr>
<td>&quot;GET_AUTHENTICATION Procedure&quot; on page 78-54</td>
<td>Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.</td>
</tr>
<tr>
<td>&quot;SET_BODY_CHARSET Procedure&quot; on page 78-55</td>
<td>Sets the character set of the response body when the media type is &quot;text&quot; but the character set is not specified in the &quot;Content-Type&quot; header.</td>
</tr>
<tr>
<td>&quot;READ_TEXT Procedure&quot; on page 78-56</td>
<td>Reads the HTTP response body in text form and returns the output in the caller-supplied buffer.</td>
</tr>
<tr>
<td>&quot;READ_LINE Procedure&quot; on page 78-57</td>
<td>Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer.</td>
</tr>
</tbody>
</table>
### Summary of UTL_HTTP Subprograms

#### Table 78–6  UTL_HTTP Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;READ_RAW Procedure&quot; on page 78-58</td>
<td>Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer.</td>
</tr>
<tr>
<td>&quot;END_RESPONSE Procedure&quot; on page 78-59</td>
<td>Ends the HTTP response. It completes the HTTP request and response.</td>
</tr>
<tr>
<td><strong>HTTP Cookies</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;GET_COOKIE_COUNT Function&quot; on page 78-60</td>
<td>Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers.</td>
</tr>
<tr>
<td>&quot;GET_COOKIES Function&quot; on page 78-60</td>
<td>Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers.</td>
</tr>
<tr>
<td>&quot;ADD_COOKIES Procedure&quot; on page 78-60</td>
<td>Adds the cookies maintained by UTL_HTTP.</td>
</tr>
<tr>
<td>&quot;CLEAR_COOKIES Procedure&quot; on page 78-61</td>
<td>Clears all cookies maintained by the UTL_HTTP package.</td>
</tr>
<tr>
<td><strong>HTTP Persistent Connections</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;GET_PERSISTENT_CONN_COUNT Function&quot; on page 78-61</td>
<td>Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers.</td>
</tr>
<tr>
<td>&quot;GET_PERSISTENT_CONNS Procedure&quot; on page 78-62</td>
<td>Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers.</td>
</tr>
<tr>
<td>&quot;CLOSE_PERSISTENT_CONN Procedure&quot; on page 78-62</td>
<td>Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session.</td>
</tr>
<tr>
<td>&quot;CLOSE_PERSISTENT_CONNS Procedure&quot; on page 78-63</td>
<td>Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session.</td>
</tr>
<tr>
<td><strong>Error Conditions</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;GET_DETAILED_SQLCODE Function&quot; on page 78-64</td>
<td>Retrieves the detailed SQLCODE of the last exception raised.</td>
</tr>
<tr>
<td>&quot;GET_DETAILED_SQLERRM Function&quot; on page 78-65</td>
<td>Retrieves the detailed SQLERRM of the last exception raised.</td>
</tr>
</tbody>
</table>
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, etc.)

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, rrps);
Parameters

Table 78–7 shows the parameters for the REQUEST function.

Table 78–7  REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Universal resource locator.</td>
</tr>
<tr>
<td>proxy</td>
<td>(Optional) Specifies a proxy server to use when making the HTTP request. See set_proxy for the full format of the proxy setting.</td>
</tr>
<tr>
<td>wallet_path</td>
<td>(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&lt;username&gt;\WALLETS, and in Unix is, for example, file:/home/&lt;username&gt;/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.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>(Optional) Specifies the password required to open the wallet.</td>
</tr>
</tbody>
</table>

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, per the 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.
REQUEST_PIECES Function

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
<HTML>
  <HEAD>
    <TITLE>Error Message</TITLE>
  </HEAD>
  <BODY>
    <H1>Fatal Error 500</H1>
    <P>
    <B>Reason:</B> Can’t locate remote host:  home.nothing.comm.
    <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;
```

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`:

```
```

**REQUEST_PIECES Function**

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.
Summary of UTL_HTTP Subprograms

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 78–8 shows the parameters for the REQUEST_PIECES function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Universal resource locator.</td>
</tr>
<tr>
<td>max_pieces</td>
<td>(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.</td>
</tr>
<tr>
<td>proxy</td>
<td>(Optional) Specifies a proxy server to use when making the HTTP request. See set_proxy for the full format of the proxy setting.</td>
</tr>
<tr>
<td>wallet_path</td>
<td>(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:/&lt;local-dir-for-client-side-wallet&gt;'. The format of wallet_path on a PC is, for example, file:\WINNT\Profiles&lt;username&gt;\WALLETS, and in Unix is, for example, file:/home/&lt;username&gt;/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.</td>
</tr>
</tbody>
</table>
REQUEST_PIECES Function

Table 78–8  REQUEST_PIECES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_password</td>
<td>(Optional) Specifies the password required to open the wallet.</td>
</tr>
</tbody>
</table>

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, per the 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
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
</P>
```
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 78–9 shows the parameters for the SET_PROXY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy (IN)</td>
<td>The proxy (host and an optional port number) to be used by the UTL_HTTP package</td>
</tr>
<tr>
<td>no_proxy_domains</td>
<td>The list of hosts and domains for which no proxy should be used for all requests.</td>
</tr>
</tbody>
</table>

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 78–10 shows the parameters for the GET_PROXY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy (OUT)</td>
<td>The proxy (host and an optional port number) currently used by the UTL_HTTP package</td>
</tr>
<tr>
<td>no_proxy_domains</td>
<td>The list of hosts and domains for which no proxy is used for all requests.</td>
</tr>
</tbody>
</table>

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 vs. 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 per site and 300 total.

**Syntax**

```sql
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 78–11 shows the parameters for the `SET_COOKIE_SUPPORT` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable (IN)</td>
<td>Sets whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)</td>
</tr>
<tr>
<td>max_cookies (IN)</td>
<td>Sets the maximum total number of cookies maintained in the current session</td>
</tr>
<tr>
<td>max_cookies_per_site (IN)</td>
<td>Sets the maximum number of cookies maintained in the current session per each Web site</td>
</tr>
</tbody>
</table>

**Usage Notes**

If you lower the maximum total number of cookies or the maximum number of cookies per 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 78-11 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

```sql
UTL_HTTP.get_cookie_support (    
   enable OUT BOOLEAN,    
   max_cookies OUT PLS_INTEGER,    
   max_cookies_per_site OUT PLS_INTEGER);
```

Parameters

Table 78–12 shows the parameters for the GET_COOKIE_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable (OUT)</td>
<td>Indicates whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)</td>
</tr>
<tr>
<td>max_cookies (OUT)</td>
<td>Indicates the maximum total number of cookies maintained in the current session</td>
</tr>
<tr>
<td>max_cookies_per_site</td>
<td>Indicates the maximum number of cookies maintained in the current session per each Web site</td>
</tr>
</tbody>
</table>

**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.
GET_FOLLOW_REDIRECT Procedure

Syntax

```
UTL_HTTP.set_follow_redirect (
    max_redirec ts IN PLS_INTEGER DEFAULT 3);
```

Parameters

Table 78–13 shows the parameters for the SET_FOLLOW_REDIRECT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_redirec ts (IN)</td>
<td>The maximum number of redirections. Set to zero to disable redirection</td>
</tr>
</tbody>
</table>

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_redirec ts OUT PLS_INTEGER);
```

Parameters

Table 78–14 shows the parameters for the GET_FOLLOW_REDIRECT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_redirec ts (OUT)</td>
<td>The maximum number of redirections for all future HTTP requests.</td>
</tr>
</tbody>
</table>

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 78–15 shows the parameters for the SET_BODY_CHARSET procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 78–16 shows the parameters for the GET_BODY_CHARSET 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 vs. 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 78-33.

Syntax

```sql
UTL_HTTP.set_persistent_conn_support (
    enable IN BOOLEAN,
    max_conns IN PLS_INTEGER DEFAULT 0);
```

Parameters

Table 78–17 shows the parameters for the SET_PERSISTENT_CONN_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable (IN)</td>
<td>Enables (set to TRUE) or disables (set to FALSE) persistent connection support</td>
</tr>
<tr>
<td>max_conns (IN)</td>
<td>Sets the maximum number of persistent connections maintained in the current session.</td>
</tr>
</tbody>
</table>

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

```sql
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/';
```
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

```sql
UTL_HTTP.get_persistent_conn_support(
```
enable OUT BOOLEAN,
max_conns OUT PLS_INTEGER);

Parameters

Table 78–18 shows the parameters for the GET_PERSISTENT_CONN_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable (OUT)</td>
<td>TRUE if persistent connection support is enabled; otherwise FALSE</td>
</tr>
<tr>
<td>max_conns (OUT)</td>
<td>the maximum number of persistent connections maintained in the current session.</td>
</tr>
</tbody>
</table>
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

```sql
UTL_HTTP.get_response_error_check (    enable OUT BOOLEAN);
```

Parameters

Table 78–20 shows the parameters for the GET_RESPONSE_ERROR_CHECK procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable (OUT)</td>
<td>TRUE if the response error check is set; otherwise FALSE</td>
</tr>
</tbody>
</table>

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

```sql
UTL_HTTP.set_detailed_excp_support (    enable IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 78–21 shows the parameters for the SET_DETAILED_EXCP_SUPPORT procedure.
GET_DETAILED_EXCP_SUPPORT Procedure

This procedure checks if the UTL_HTTP package will raise a detailed exception or not.

Syntax

```sql
UTL_HTTP.get_detailed_excp_support (enable OUT BOOLEAN);
```

Parameters

Table 78–22 shows the parameters for the GET_DETAILED_EXCP_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE if UTL_HTTP raises a detailed exception; otherwise FALSE</td>
</tr>
</tbody>
</table>

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
SET_TRANSFER_TIMEOUT Procedure

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 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 78–23 shows the parameters for the SET_WALLET procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The directory path that contains the Oracle wallet. The format is file:&lt;directory-path&gt;. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles&lt;username&gt;\WALLETS, and in Unix is, for example, file:/home/&lt;username&gt;/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.</td>
</tr>
<tr>
<td>password</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 78–24 shows the parameters for the SET_TRANSFER_TIMEOUT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEOUT (IN)</td>
<td>The network transfer timeout value in seconds.</td>
</tr>
</tbody>
</table>

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 78–25 shows the parameters for the GET_TRANSFER_TIMEOUT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEOUT (OUT)</td>
<td>The network transfer timeout value in seconds.</td>
</tr>
</tbody>
</table>

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 function 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.
SET_HEADER Procedure

Syntax

```sql
UTL_HTTP.begin_request (
    url IN VARCHAR2,
    method IN VARCHAR2 DEFAULT 'GET',
    http_version IN VARCHAR2 DEFAULT NULL)
RETURN req;
```

Parameters

Table 78–26 shows the parameters for the BEGIN_REQUEST function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url (IN)</td>
<td>The URL of the HTTP request</td>
</tr>
<tr>
<td>method (IN)</td>
<td>The method performed on the resource identified by the URL</td>
</tr>
<tr>
<td>http_version (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 84, "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.
Summary of UTL_HTTP Subprograms

Syntax

```
UTL_HTTP.set_header (  
    r IN OUT NOCOPY req,  
    name IN VARCHAR2,  
    value IN VARCHAR2);
```

Parameters

Table 78–26 shows the parameters for the SET_HEADER procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>name (IN)</td>
<td>The name of the HTTP request header</td>
</tr>
<tr>
<td>value (IN)</td>
<td>The value of the HTTP request header</td>
</tr>
</tbody>
</table>

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.
SET_COOKIE_SUPPPORT Procedure

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 78–28 shows the parameters for the SET_AUTHENTICATION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>username (IN)</td>
<td>The username for the HTTP authentication</td>
</tr>
<tr>
<td>password (IN)</td>
<td>The password for the HTTP authentication</td>
</tr>
<tr>
<td>scheme (IN)</td>
<td>The HTTP authentication scheme. The default, BASIC, denotes the HTTP Basic Authentication scheme.</td>
</tr>
<tr>
<td>for_proxy (IN)</td>
<td>Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is FALSE.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_HTTP.set_cookie_support(
    r IN OUT NOCOPY req,
    enable IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 78–28 shows the parameters for the SET_COOKIE_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>enable (IN)</td>
<td>Set enable to TRUE to enable HTTP cookie support; FALSE to disable</td>
</tr>
</tbody>
</table>

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 78-11 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.
SET_BODY_CHARSET Procedure

Use this procedure to change the maximum number of redirections a request inherits from the session default setting.

Syntax

```plsql
UTL_HTTP.set_follow_redirect(
    r IN OUT NOCOPY req,
    max_redirects IN PLS_INTEGER DEFAULT 3);
```

Parameters

Table 78–30 shows the parameters for the SET_FOLLOW_REDIRECT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>max_redirects (IN)</td>
<td>The maximum number of redirects. Set to zero to disable redirects.</td>
</tr>
</tbody>
</table>

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. Per 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

```plsql
UTL_HTTP.set_body_charset(
    r IN OUT NOCOPY req,
    charset IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 78–30 shows the parameters for the SET_BODY_CHARSET procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>charset (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 per 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 78-46.

Syntax

```
UTL_HTTP.set_persistent_conn_support(
    r IN OUT NOCOPY req,
    enable IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 78–32 shows the parameters for the SET_PERSISTENT_CONN_SUPPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>enable (IN)</td>
<td>TRUE to keep the network connection persistent. FALSE otherwise.</td>
</tr>
</tbody>
</table>

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 LOOP;

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**

```sql
UTL_HTTP.write_text(
  r IN OUT NOCOPY req,
  data IN VARCHAR2);
```
WRITE_LINE Procedure

Parameters

Table 78–33 shows the parameters for the WRITE_TEXT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data (IN)</td>
<td>The text data to send in the HTTP request body</td>
</tr>
</tbody>
</table>

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 multi-byte 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

```sql
UTL_HTTP.write_line(
    r IN OUT NOCOPY req,
    data IN VARCHAR2);
```

Parameters

Table 78–34 shows the parameters for the WRITE_LINE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data (IN)</td>
<td>The text line to send in the HTTP request body</td>
</tr>
</tbody>
</table>

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 multi-byte 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**

```sql
UTL_HTTP.write_raw(
   r IN OUT NOCOPY req,
   data IN RAW);
```

**Parameters**

Table 78–35 shows the parameters for the WRITE_RAW procedure.

**Table 78–35  WRITE_RAW Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data (IN)</td>
<td>The binary data to send in the HTTP request body</td>
</tr>
</tbody>
</table>

**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**

```sql
UTL_HTTP.end_request (  
    r  IN OUT NOCOPY req);  
```

**Parameters**

Table 78–36 shows the parameters for the END_REQUEST procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
</tbody>
</table>

**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**

```sql
UTL_HTTP.get_response (  
    r  IN OUT NOCOPY req)  
RETURN resp;  
```
GET_HEADER_COUNT Function

Parameters

Table 78–37 shows the parameters for the GET_RESPONSE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response</td>
</tr>
</tbody>
</table>

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 78–38 shows the parameters for the GET_HEADER_COUNT function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response</td>
</tr>
</tbody>
</table>

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<sup>th</sup> HTTP response header name and value returned in the response.
Summary of UTL_HTTP Subprograms

Syntax

```sql
UTL_HTTP.get_header ( 
    r  IN OUT NOCOPY resp,  
    n  IN PLS_INTEGER,  
    name OUT NOCOPY VARCHAR2,  
    value OUT NOCOPY VARCHAR2);  
```

Parameters

Table 78–39 shows the parameters for the GET_HEADER procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>n (IN)</td>
<td>The n&lt;sup&gt;th&lt;/sup&gt; header to return.</td>
</tr>
<tr>
<td>name (OUT)</td>
<td>The name of the HTTP response header.</td>
</tr>
<tr>
<td>value (OUT)</td>
<td>The value of the HTTP response header.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_HTTP.get_header_by_name( 
    r  IN OUT NOCOPY resp,  
    name IN VARCHAR2,  
    value OUT NOCOPY VARCHAR2,  
    n  IN PLS_INTEGER DEFAULT 1);  
```
GET_AUTHENTICATION Procedure

Parameters

Table 78–40 shows the parameters for the GET_HEADER_BY_NAME procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>n (IN)</td>
<td>The n&lt;sup&gt;th&lt;/sup&gt; occurrence of an HTTP response header by the specified name to return. The default is 1.</td>
</tr>
<tr>
<td>name (IN)</td>
<td>The name of the HTTP response header for which the value is to return</td>
</tr>
<tr>
<td>value (OUT)</td>
<td>The value of the HTTP response header.</td>
</tr>
</tbody>
</table>

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

```lisp
UTL_HTTP.get_authentication(
    r IN OUT NOCOPY resp,
    scheme OUT VARCHAR2,
    realm OUT VARCHAR2,
    for_proxy IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 78–41 shows the parameters for the GET_AUTHENTICATION procedure.
Summary of UTL_HTTP Subprograms

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. Per 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**

```sql
UTL_HTTP.set_body_charset(
    r   IN OUT NOCOPY resp,
    charset   IN VARCHAR2 DEFAULT NULL);
```

**Table 78–41 GETAUTHENTICATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>scheme (OUT)</td>
<td>The scheme for the required HTTP authentication</td>
</tr>
<tr>
<td>realm (OUT)</td>
<td>The realm for the required HTTP authentication</td>
</tr>
<tr>
<td>for_proxy (IN)</td>
<td>Returns the HTTP authentication information required for the access to the HTTP proxy server instead of the Web server? Default is FALSE.</td>
</tr>
</tbody>
</table>
READ_TEXT Procedure

Parameters

Table 78–42 shows the parameters for the SET_BODY_CHARSET procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>charset (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_HTTP.read_text(
    r IN OUT NOCOPY resp,
    data OUT NOCOPY VARCHAR2,
    len IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 78–43 shows the parameters for the READ_TEXT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>data (OUT)</td>
<td>The HTTP response body in text form</td>
</tr>
<tr>
<td>len (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 multi-byte character is found at the end of the response body, **read_text** stops reading and returns all the complete multi-byte 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 multi-byte character can be read as binary by the **read_raw** procedure. If a partial multi-byte 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.

**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**

```sql
UTL_HTTP.read_line(
    r IN OUT NOCOPY resp,
    data OUT NOCOPY VARCHAR2,
    remove_crlf IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

**Table 78–44** shows the parameters for the **READ_LINE** procedure.
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 multi-byte character is found at the end of the response body, read_line stops reading and returns all the complete multi-byte 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 multi-byte character can be read as binary by the read_raw procedure. If a partial multi-byte 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.

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

```sql
UTL_HTTP.read_raw(
    r IN OUT NOCOPY resp,
    data OUT NOCOPY RAW,
    len IN PLS_INTEGER DEFAULT NULL);
```
Parameters

Table 78–45 shows the parameters for the READ_RAW procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (IN/OUT)</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>data (OUT)</td>
<td>The HTTP response body in binary form</td>
</tr>
<tr>
<td>len (IN)</td>
<td>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</td>
</tr>
</tbody>
</table>

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

```sql
UTL_HTTP.end_response (  
    r IN OUT NOCOPY resp);
```

Parameters

Table 78–46 shows the parameters for the END_RESPONSE procedure.
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 78-47 shows the parameters for the GET_COOKIES procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookies</td>
<td>The cookies returned</td>
</tr>
</tbody>
</table>

ADD_COOKIES Procedure
This procedure adds the cookies maintained by UTL_HTTP.
Summary of UTL_HTTP Subprograms

**Syntax**

```
UTL_HTTP.add_cookies (  
    cookies IN cookie_table);
```

**Parameters**

*Table 78–48* shows the parameters for the *ADD_COOKIES* procedure.

*Table 78–48  ADD_COOKIES Procedure Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookies (IN/OUT)</td>
<td>The cookies to be added</td>
</tr>
</tbody>
</table>

**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;
```
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**

```sql
UTL_HTTP.get_persistent_conns (  
    connections IN OUT NOCOPY connection_table);
```

**Parameters**

Table 78-49 shows the parameters for the GET_PERSISTENT_CONNS procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connections</td>
<td>The network connections kept persistent</td>
</tr>
</tbody>
</table>

**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**

```sql
UTL_HTTP.close_persistent_conn (  
    conn IN connection);
```
Parameters

Table 78–50 shows the parameters for the CLOSE_PERSISTENT_CONN procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn (IN)</td>
<td>The HTTP persistent connection to close</td>
</tr>
</tbody>
</table>

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:

```sql
utl_http.close_persistent_conns(host => 'foobar');
```

And the following call to the procedure closes all persistent connections via the proxy www-proxy at TCP/IP port 80:

```sql
utl_http.close_persistent_conns(proxy_host => 'foobar',
                                proxy_port => 80);
```

And the following call to the procedure closes all persistent connections:

```sql
utl_http.close_persistent_conns;
```

Syntax

```sql
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);
```
Table 78–51 shows the parameters for the CLOSE_PERSISTENT_CONNS procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host (IN)</td>
<td>The host for which persistent connections are to be closed</td>
</tr>
<tr>
<td>port (IN)</td>
<td>The port number for which persistent connections are to be closed</td>
</tr>
<tr>
<td>proxy_host (IN)</td>
<td>The proxy host for which persistent connections are to be closed</td>
</tr>
<tr>
<td>proxy_port (IN)</td>
<td>The proxy port for which persistent connections are to be closed</td>
</tr>
<tr>
<td>ssl (IN)</td>
<td>Close persistent SSL connection</td>
</tr>
</tbody>
</table>

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
GET_DETAILED_SQLERRM Function

This function retrieves the detailed SQLERRM of the last exception raised.

Syntax

```plsql
UTL_HTTP.get_detailed_sqlerrm
RETURN VARCHAR2;
```
UTL_INADDR provides 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
- Summary of UTL_INADDR Subprograms
Exceptions

The exception raised by the Internet Address package appears in Table 79–1.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN_HOST</td>
<td>The host is unknown.</td>
</tr>
</tbody>
</table>

Summary of UTL_INADDR Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;get_host_name Function&quot; on page 79-2</td>
<td>Retrieves the name of the local or remote host given its IP address.</td>
</tr>
<tr>
<td>&quot;get_host_address Function&quot; on page 79-3</td>
<td>Retrieves the IP address of the local or remote host given its name.</td>
</tr>
</tbody>
</table>

get_host_name Function

This function retrieves the name of the local or remote host given its IP address.

Syntax

```sql
UTL_INADDR.GET_HOST_NAME (  
ip IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```sql
UTL_INADDR.GET_HOST_ADDRESS (host IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host (IN)</td>
<td>The name of the host to retrieve the IP address. If host is NULL, this function returns the IP address of the local host.</td>
</tr>
</tbody>
</table>
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
- Summary of UTL_RAW Subprograms
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CAST_FROM_BINARY_INTEGER Function&quot; on page 80-3</td>
<td>Returns the binary representation of a BINARY_INTEGER (in RAW).</td>
</tr>
<tr>
<td>&quot;CAST_FROM_NUMBER Function&quot; on page 80-4</td>
<td>Returns the binary representation of a NUMBER (in RAW).</td>
</tr>
<tr>
<td>&quot;CAST_TO_BINARY_INTEGER Function&quot; on page 80-4</td>
<td>Casts the binary representation of a BINARY_INTEGER (in RAW) into a BINARY_INTEGER.</td>
</tr>
<tr>
<td>&quot;CAST_TO_NUMBER Function&quot; on page 80-5</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;CAST_TO_RAW Function&quot; on page 80-5</td>
<td>Converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes.</td>
</tr>
<tr>
<td>&quot;CAST_TO_VARCHAR2 Function&quot; on page 80-6</td>
<td>Converts a RAW represented using n data bytes into VARCHAR2 with n data bytes.</td>
</tr>
<tr>
<td>&quot;CONCAT Function&quot; on page 80-7</td>
<td>Concatenates up to 12 RAWs into a single RAW.</td>
</tr>
<tr>
<td>&quot;LENGTH Function&quot; on page 80-8</td>
<td>Returns the length in bytes of a RAW r.</td>
</tr>
<tr>
<td>&quot;SUBSTR Function&quot; on page 80-9</td>
<td>Returns len bytes, starting at pos from RAW r.</td>
</tr>
<tr>
<td>&quot;TRANSLATE Function&quot; on page 80-10</td>
<td>Translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set.</td>
</tr>
<tr>
<td>&quot;TRANSLITERATE Function&quot; on page 80-12</td>
<td>Converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set.</td>
</tr>
</tbody>
</table>
Table 80–1  UTL_RAW Subprograms (Cont.)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;OVERLAY Function&quot; on page 80-14</td>
<td>Overlays the specified portion of target RAW with overlay RAW, starting from byte position pos of target and proceeding for len bytes.</td>
</tr>
<tr>
<td>&quot;COPIES Function&quot; on page 80-15</td>
<td>Returns n copies of r concatenated together.</td>
</tr>
<tr>
<td>&quot;XRANGE Function&quot; on page 80-16</td>
<td>Returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte.</td>
</tr>
<tr>
<td>&quot;REVERSE Function&quot; on page 80-17</td>
<td>Reverses a byte sequence in RAW r from end to end.</td>
</tr>
<tr>
<td>&quot;COMPARE Function&quot; on page 80-18</td>
<td>Compares RAW r1 against RAW r2.</td>
</tr>
<tr>
<td>&quot;CONVERT Function&quot; on page 80-19</td>
<td>Converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.</td>
</tr>
<tr>
<td>&quot;BIT_AND Function&quot; on page 80-20</td>
<td>Performs bitwise logical &quot;and&quot; of the values in RAW r1 with RAW r2 and returns the &quot;anded&quot; result RAW.</td>
</tr>
<tr>
<td>&quot;BIT_OR Function&quot; on page 80-21</td>
<td>Performs bitwise logical &quot;or&quot; of the values in RAW r1 with RAW r2 and returns the &quot;or'd&quot; result RAW.</td>
</tr>
<tr>
<td>&quot;BIT_XOR Function&quot; on page 80-22</td>
<td>Performs bitwise logical &quot;exclusive or&quot; of the values in RAW r1 with RAW r2 and returns the &quot;xor’d&quot; result RAW.</td>
</tr>
<tr>
<td>&quot;BIT_COMPLEMENT Function&quot; on page 80-23</td>
<td>Performs bitwise logical &quot;complement&quot; of the values in RAW r and returns the &quot;complement'ed&quot; result RAW.</td>
</tr>
</tbody>
</table>

CAST_FROM_BINARY_INTEGER Function

This function returns the binary representation of a BINARY_INTEGER (in RAW).

Syntax

```sql
UTL_RAW.CAST_FROM_BINARY_INTEGER (n IN BINARY_INTEGER, endianess in PLS_INTEGER DEFAULT BIG_ENDIAN) RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(cast_from_binary_integer, WNDS, RNDS, WNPS, RNPS);
```
CAST_FROM_NUMBER Function

Parameters

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

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

```plaintext
pragma restrict_references(cast_to_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

- `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

```plaintext
UTL_RAW.CAST_TO_NUMBER (r IN RAW, include_length IN BOOLEAN) RETURN NUMBER;
```

Pragmas

```plaintext
pragma restrict_references(cast_to_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

- `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.
CAST_TO_VARCHAR2 Function

Syntax

```sql
UTL_RAW.CAST_TO_RAW (  
    c IN VARCHAR2)  
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>VARCHAR2 to be changed to a RAW.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field.</td>
</tr>
<tr>
<td>NULL</td>
<td>If c input parameter was NULL.</td>
</tr>
</tbody>
</table>

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 NLS character set is used for the characters within that VARCHAR2.

Syntax

```sql
UTL_RAW.CAST_TO_VARCHAR2 (  
    r IN RAW)  
RETURN VARCHAR2;
```
Pragmas

```sql
pragma restrict_references(cast_to_varchar2, WDNS, RNDS, WNPS, RNPS);
```

Parameters

**Table 80–4 CAST_TO_VARCHAR2 Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>RAW (without leading length field) to be changed to a VARCHAR2.</td>
</tr>
</tbody>
</table>

Returns

**Table 80–5 CAST_TO_VARCHAR2 Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Containing having the same data as the input RAW.</td>
</tr>
<tr>
<td>NULL</td>
<td>If <code>r</code> input parameter was NULL.</td>
</tr>
</tbody>
</table>

**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

```sql
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;
```
LENGTH Function

Pragmas

```
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);
```

Parameters

\[ r_1 \ldots r_{12} \] are the \texttt{RAW} items to concatenate.

Returns

<table>
<thead>
<tr>
<th>Table 80–6 CONCAT Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return</strong></td>
</tr>
<tr>
<td>\texttt{RAW}</td>
</tr>
</tbody>
</table>

Errors

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a \texttt{RAW}, which is 32767 bytes.

LENGTH Function

This function returns the length in bytes of a \texttt{RAW} \( r \).

Syntax

```
UTL_RAW.LENGTH ( 
  r IN RAW) 
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Table 80–7 LENGTH Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>( r )</td>
</tr>
</tbody>
</table>
Returns

**Table 80-8  LENGTH Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Equal to the current length of the RAW.</td>
</tr>
</tbody>
</table>

**SUBSTR Function**

This function returns len bytes, starting at pos from RAW r.

**Syntax**

```plsql
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 backwards 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 80-9  SUBSTR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW byte-string from which a portion is extracted.</td>
</tr>
<tr>
<td>pos</td>
<td>The byte position in r at which to begin extraction.</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes from pos to extract from r (optional).</td>
</tr>
</tbody>
</table>
TRANSLATE Function

Defaults and Optional Parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>Position pos through to the end of r.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>portion of r</td>
<td>Beginning at pos for len bytes long.</td>
</tr>
<tr>
<td>NULL</td>
<td>R input parameter was NULL.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either pos = 0 or len &lt; 0</td>
</tr>
</tbody>
</table>

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.
Summary of UTL_RAW Subprograms

---

**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**

```plaintext
UTL_RAW.TRANSLATE (  
   r    IN RAW,  
   from_set IN RAW,  
   to_set  IN RAW)  
RETURN RAW;
```

**Pragmas**

```plaintext
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW source byte-string to be translated.</td>
</tr>
<tr>
<td>from_set</td>
<td>RAW byte-codes to be translated, if present in r.</td>
</tr>
<tr>
<td>to_set</td>
<td>RAW byte-codes to which corresponding from_str bytes are translated.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Translated byte-string.</td>
</tr>
</tbody>
</table>
TRANSLITERATE Function

Errors

Table 80–15 TRANSLATE Function Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- r is NULL and/or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- from_set is NULL and/or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- to_set is NULL and/or has 0 length</td>
</tr>
</tbody>
</table>

TRANSLITERATE Function

This function converts the bytes in the input RAW \( r \) according to the bytes in the transliteration RAWs \( \text{from\_set} \) and \( \text{to\_set} \). Successive bytes in \( r \) are looked up in the \( \text{from\_set} \), 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 \( \text{to\_set} \), or the pad byte when no correspondence exists.

Bytes in \( r \), but undefined in \( \text{from\_set} \), are copied to the result. Only the first (leftmost) occurrence of a byte in \( \text{from\_set} \) is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as \( r \).

If the \( \text{to\_set} \) is shorter than the \( \text{from\_set} \), then the pad byte is placed in the result RAW when a selected \( \text{from\_set} \) byte has no corresponding \( \text{to\_set} \) byte (as if the \( \text{to\_set} \) were extended to the same length as the \( \text{from\_set} \) with pad bytes).

Note: Difference from TRANSLATE:

- \( r \) bytes undefined in \( \text{to\_set} \) are padded.
- Result RAW is always same length as input RAW \( r \).

Syntax

```sql
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;
```
Summary of UTL_RAW Subprograms

Pragmas

```
pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

**Table 80–16 TRANSLITERATE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>RAW input byte-string to be converted.</td>
</tr>
<tr>
<td><code>from_set</code></td>
<td>RAW byte-codes to be converted, if present in <code>r</code> (any length).</td>
</tr>
<tr>
<td><code>to_set</code></td>
<td>RAW byte-codes to which corresponding <code>from_set</code> bytes are converted (any length).</td>
</tr>
<tr>
<td><code>pad</code></td>
<td>1 byte used when <code>to_set</code> is shorter than the <code>from_set</code>.</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

**Table 80–17 TRANSLITERATE Function Optional Parameters**

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_set</code></td>
<td>x'00 through x'ff.</td>
</tr>
<tr>
<td><code>to_set</code></td>
<td>To the NULL string and effectively extended with <code>pad</code> to the length of <code>from_set</code> as necessary.</td>
</tr>
<tr>
<td><code>pad</code></td>
<td>x'00'.</td>
</tr>
</tbody>
</table>

Returns

**Table 80–18 TRANSLITERATE Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Converted byte-string.</td>
</tr>
</tbody>
</table>

Errors

**Table 80–19 TRANSLITERATE Function Errors**

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td><code>r</code> is NULL and/or has 0 length.</td>
</tr>
</tbody>
</table>
**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**

```sql
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>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay_str</td>
<td>Byte-string used to overlay target.</td>
</tr>
<tr>
<td>target</td>
<td>Byte-string which is to be overlayed.</td>
</tr>
<tr>
<td>pos</td>
<td>Position in target (numbered from 1) to start overlay.</td>
</tr>
<tr>
<td>len</td>
<td>The number of target bytes to overlay.</td>
</tr>
<tr>
<td>pad</td>
<td>Pad byte used when overlay len exceeds overlay length or pos exceeds target length.</td>
</tr>
</tbody>
</table>
Defaults and Optional Parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>1</td>
</tr>
<tr>
<td>len</td>
<td>To the length of overlay</td>
</tr>
<tr>
<td>pad</td>
<td>x'00'</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The target byte_string overlayed as specified.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- Overlay is NULL and/or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- Target is missing or undefined</td>
</tr>
<tr>
<td></td>
<td>- Length of target exceeds maximum length of a RAW</td>
</tr>
<tr>
<td></td>
<td>- len &lt; 0</td>
</tr>
<tr>
<td></td>
<td>- pos &lt; 1</td>
</tr>
</tbody>
</table>

COPIES Function

This function returns \(n\) copies of \(r\) concatenated together.

Syntax

```sql
UTL_RAW.COPIES (
    r IN RAW,
    n IN NUMBER)
RETURN RAW;
```
XRANGE Function

Pragmas

```sql
pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>RAW to be copied</td>
</tr>
<tr>
<td><code>n</code></td>
<td>Number of times to copy the RAW (must be positive).</td>
</tr>
</tbody>
</table>

Returns

This returns the RAW copied `n` times.

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- <code>r</code> is missing, NULL and/or 0 length</td>
</tr>
<tr>
<td></td>
<td>- <code>n</code> &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- Length of result exceeds maximum length of a RAW</td>
</tr>
</tbody>
</table>

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

```sql
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 80–26 XRANGE Function Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_byte</td>
<td>Beginning byte-code value of resulting sequence.</td>
</tr>
<tr>
<td>end_byte</td>
<td>Ending byte-code value of resulting sequence.</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

- start_byte - x’00’
- start_byte - x’00’
- end_byte   - x’FF’

Returns

**Table 80–27 XRANGE Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing succession of 1-byte hexadecimal encodings.</td>
</tr>
</tbody>
</table>

**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);
```
COMPARE Function

Parameters

Table 80–28  REVERSE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW to reverse.</td>
</tr>
</tbody>
</table>

Returns

Table 80–29  REVERSE Function Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;reverse&quot; of r.</td>
</tr>
</tbody>
</table>

Errors

Table 80–30  REVERSE Function Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>R is NULL and/or has 0 length.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_RAW.COMPARE ( 
    r1 IN RAW, 
    r2 IN RAW, 
    pad IN RAW DEFAULT NULL) 
RETURN NUMBER;
```

Pragmas

```sql
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

80-18  Supplied PL/SQL Packages and Types Reference
Summary of UTL_RAW Subprograms

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1st RAW to be compared, may be NULL and/or 0 length.</td>
</tr>
<tr>
<td>r2</td>
<td>2nd RAW to be compared, may be NULL and/or 0 length.</td>
</tr>
<tr>
<td>pad</td>
<td>Byte to extend whichever of r1 or r2 is shorter.</td>
</tr>
</tbody>
</table>

Defaults and optional parameters

pad = '00'

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Equals 0 if RAW byte strings are both NULL or identical; or, Equals position (numbered from 1) of the first mismatched byte.</td>
</tr>
</tbody>
</table>

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, WND5, RND5, WNPS, RNPS);
```
BIT_AND Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW byte-string to be converted.</td>
</tr>
<tr>
<td>to_charset</td>
<td>Name of NLS character set to which r is converted.</td>
</tr>
<tr>
<td>from_charset</td>
<td>Name of NLS character set in which r is supplied.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Byte string r converted according to the specified character sets.</td>
</tr>
</tbody>
</table>

Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- r missing, NULL, and/or 0 length</td>
</tr>
<tr>
<td></td>
<td>- from_charset or to_charset missing, NULL, and/or 0 length</td>
</tr>
<tr>
<td></td>
<td>- from_charset or to_charset names invalid or unsupported</td>
</tr>
</tbody>
</table>

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

```
UTIL_RAW.BIT_AND (  
    r1 IN RAW,  
```

80-20   Supplied PL/SQL Packages and Types Reference
Pragmas

```
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

Parameters

**Table 80–36 BIT_AND Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>RAW to &quot;and&quot; with r2.</td>
</tr>
<tr>
<td>r2</td>
<td>RAW to &quot;and&quot; with r1.</td>
</tr>
</tbody>
</table>

Returns

**Table 80–37 BIT_AND Function Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;and&quot; of r1 and r2.</td>
</tr>
<tr>
<td>NULL</td>
<td>Either r1 or r2 input parameter was NULL.</td>
</tr>
</tbody>
</table>

**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;
```
BIT_XOR Function

Pragmas

```sql
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>RAW to &quot;or&quot; with r2.</td>
</tr>
<tr>
<td>r2</td>
<td>RAW to &quot;or&quot; with r1.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;or&quot; of r1 and r2.</td>
</tr>
<tr>
<td>NULL</td>
<td>Either r1 or r2 input parameter was NULL.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_RAW.BIT_XOR ( 
    r1 IN RAW, 
    r2 IN RAW) 
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);
```
Summary of UTL_RAW Subprograms

Parameters

<table>
<thead>
<tr>
<th>Table 80–40 BIT_XOR Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Table 80–41 BIT_XOR Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
</tr>
<tr>
<td>RAW</td>
</tr>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>

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

```plaintext
UTL_RAW.BIT_COMPLEMENT ( r IN RAW) RETURN RAW;
```

Pragmas

```plaintext
pragma restrict_references(bit_complement, WNDS, RNDS, WNP3, RNP3);
```

Parameters

<table>
<thead>
<tr>
<th>Table 80–42 BIT_COMPLEMENT Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>r</td>
</tr>
</tbody>
</table>
### BIT_COMPLEMENT Function

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The &quot;complement&quot; of r1.</td>
</tr>
<tr>
<td>NULL</td>
<td>If r input parameter was NULL.</td>
</tr>
</tbody>
</table>
Oracle8i 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
- Datatypes, Exceptions, and Security for UTL_REF
- Summary of UTL_REF Subprograms
Requirements

The procedural option is needed to use this package. 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.

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:

```sql
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:

```sql
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.
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`.

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errnum == 942</code></td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td><code>errnum == 1031</code></td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td><code>errnum == 8177</code></td>
<td>Unable to serialize, if in a serializable transaction.</td>
</tr>
<tr>
<td><code>errnum == 60</code></td>
<td>Deadlock detected.</td>
</tr>
<tr>
<td><code>errnum == 1403</code></td>
<td>No data found (if the <code>REF</code> is null, etc.).</td>
</tr>
</tbody>
</table>

**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 81–2  UTL_REF Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SELECT_OBJECT Procedure&quot; on page 81-4</td>
<td>Selects an object given a reference.</td>
</tr>
<tr>
<td>&quot;LOCK_OBJECT Procedure&quot; on page 81-5</td>
<td>Locks an object given a reference.</td>
</tr>
<tr>
<td>&quot;UPDATE_OBJECT Procedure&quot; on page 81-6</td>
<td>Updates an object given a reference.</td>
</tr>
<tr>
<td>&quot;DELETE_OBJECT Procedure&quot; on page 81-6</td>
<td>Deletes an object given a reference.</td>
</tr>
</tbody>
</table>

**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:

```sql
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**

```sql
UTL_REF.SELECT_OBJECT (  
    reference IN REF "<typename>",  
    object IN OUT "<typename>"  
);
```

**Parameters**

**Table 81–3  SELECT_OBJECT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference to the object to select or retrieve.</td>
</tr>
<tr>
<td>object</td>
<td>The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.</td>
</tr>
</tbody>
</table>
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:

```sql
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

```sql
UTL_REF.LOCK_OBJECT (
  reference IN REF "<typename>";
)

UTL_REF.LOCK_OBJECT (
  reference IN REF "<typename>",
  object IN OUT "<typename>");
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to lock.</td>
</tr>
<tr>
<td>object</td>
<td>The PL/SQL variable that stores the locked object. This variable should be of the same object type as the locked object.</td>
</tr>
</tbody>
</table>

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:

```sql
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**

```sql
UTL_REF.UPDATE_OBJECT ( 
    reference IN REF "<typename>",
    object IN "<typename>";
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to update.</td>
</tr>
<tr>
<td>object</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

```sql
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

```sql
UTL_REF.DELETE_OBJECT (
    reference IN REF "<typename>";
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to delete.</td>
</tr>
</tbody>
</table>

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.

```sql
... 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),
```
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  
ernum := SQLCODE;  
ermsg := SUBSTR(SQLERRM, 1, 200);
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.

Also, note that UTL_SMTP provides API for SMTP communication as specified in RFC821. The package does not provide API to format the content of the message according to RFC 822 (for example, setting the subject of an electronic mail). It is the user’s responsibility to format the message appropriately.

This chapter discusses the following topics:

- Example
- Exceptions, Limitations, and Reply Codes
Summary of UTL_SMTP Subprograms

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/
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.

```sql
UTL_SMTP.send_mail (  
    sender IN VARCHAR2,  
    recipient IN VARCHAR2,  
    message IN VARCHAR2)  
IS  
  mailhost VARCHAR2(30) := 'mailhost.mydomain.com';  
  mail_conn utl_smtp.connection;
BEGIN  
  mail_conn := utl_smtp.open_connection(mailhost, 25);  
  utl_smtp.helo(mail_conn, mailhost);  
  utl_smtp.mail(mail_conn, sender);  
  utl_smtp.rcpt(mail_conn, recipient);  
  utl_smtp.data(mail_conn, message);  
  utl_smtp.quit(mail_conn);  
EXCEPTION  
  WHEN OTHERS THEN    
    -- Handle the error
END;
```

Exceptions, Limitations, and Reply Codes

Exceptions

Table 82–1 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>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_OPERATION</td>
<td>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().</td>
</tr>
<tr>
<td>TRANSIENT_ERROR</td>
<td>Raised when receiving a reply code in 400 range.</td>
</tr>
<tr>
<td>PERMANENT_ERROR</td>
<td>Raised when receiving a reply code in 500 range.</td>
</tr>
</tbody>
</table>
Exceptions, Limitations, and Reply Codes

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 82–2  SMTP Size Limitation

<table>
<thead>
<tr>
<th>Element</th>
<th>Size Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>The maximum total length of a user name is 64 characters.</td>
</tr>
<tr>
<td>domain</td>
<td>The maximum total length of a domain name or number is 64 characters.</td>
</tr>
<tr>
<td>path</td>
<td>The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).</td>
</tr>
<tr>
<td>command line</td>
<td>The maximum total length of a command line including the command word and the <code>&lt;CRLF&gt;</code> is 512 characters.</td>
</tr>
<tr>
<td>reply line</td>
<td>The maximum total length of a reply line including the reply code and the <code>&lt;CRLF&gt;</code> is 512 characters.</td>
</tr>
<tr>
<td>text line</td>
<td>The maximum total length of a text line including the <code>&lt;CRLF&gt;</code> is 1000 characters (but not counting the leading dot duplicated for transparency).</td>
</tr>
<tr>
<td>recipients buffer</td>
<td>The maximum total number of recipients that must be buffered is 100 recipients.</td>
</tr>
</tbody>
</table>

Reply Codes
The following is a list of the SMTP reply codes.

### Table 82–3  SMTP Reply Codes

<table>
<thead>
<tr>
<th>Reply Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>211</td>
<td>System status, or system help reply</td>
</tr>
<tr>
<td>214</td>
<td>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]</td>
</tr>
<tr>
<td>220</td>
<td><code>&lt;domain&gt;</code> Service ready</td>
</tr>
<tr>
<td>221</td>
<td><code>&lt;domain&gt;</code> Service closing transmission channel</td>
</tr>
<tr>
<td>250</td>
<td>Requested mail action okay, completed</td>
</tr>
<tr>
<td>251</td>
<td>User not local; will forward to <code>&lt;forward-path&gt;</code></td>
</tr>
</tbody>
</table>
Table 82–3  SMTP Reply Codes

<table>
<thead>
<tr>
<th>Reply Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>252</td>
<td>OK, pending messages for node <code>&lt;node&gt;</code> started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.</td>
</tr>
<tr>
<td>253</td>
<td>OK, <code>&lt;messages&gt;</code> pending messages for node <code>&lt;node&gt;</code> started</td>
</tr>
<tr>
<td>354</td>
<td>Start mail input; end with <code>&lt;CRLF&gt;</code>..&lt;CRLF&gt;</td>
</tr>
<tr>
<td>355</td>
<td>Octet-offset is the transaction offset</td>
</tr>
<tr>
<td>421</td>
<td><code>&lt;domain&gt;</code> Service not available, closing transmission channel (This may be a reply to any command if the service knows it must shut down.)</td>
</tr>
<tr>
<td>450</td>
<td>Requested mail action not taken: mailbox unavailable [for example, mailbox busy]</td>
</tr>
<tr>
<td>451</td>
<td>Requested action aborted: local error in processing</td>
</tr>
<tr>
<td>452</td>
<td>Requested action not taken: insufficient system storage</td>
</tr>
<tr>
<td>453</td>
<td>You have no mail.</td>
</tr>
<tr>
<td>454</td>
<td>TLS not available due to temporary reason. Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>458</td>
<td>Unable to queue messages for node <code>&lt;node&gt;</code></td>
</tr>
<tr>
<td>459</td>
<td>Node <code>&lt;node&gt;</code> not allowed: reason</td>
</tr>
<tr>
<td>500</td>
<td>Syntax error, command unrecognized (This may include errors such as command line too long.)</td>
</tr>
<tr>
<td>501</td>
<td>Syntax error in parameters or arguments</td>
</tr>
<tr>
<td>502</td>
<td>Command not implemented</td>
</tr>
<tr>
<td>503</td>
<td>Bad sequence of commands</td>
</tr>
<tr>
<td>504</td>
<td>Command parameter not implemented</td>
</tr>
<tr>
<td>521</td>
<td><code>&lt;Machine&gt;</code> does not accept mail.</td>
</tr>
<tr>
<td>530</td>
<td>Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>534</td>
<td>Authentication mechanism is too weak.</td>
</tr>
<tr>
<td>538</td>
<td>Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>550</td>
<td>Requested action not taken: mailbox unavailable [for example, mailbox not found, no access]</td>
</tr>
</tbody>
</table>
Summary of UTL_SMTP Subprograms

Table 82–3  SMTP Reply Codes

<table>
<thead>
<tr>
<th>Reply Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>551</td>
<td>User not local; please try &lt;forward-path&gt;</td>
</tr>
<tr>
<td>552</td>
<td>Requested mail action aborted: exceeded storage allocation</td>
</tr>
<tr>
<td>553</td>
<td>Requested action not taken: mailbox name not allowed [for example, mailbox syntax incorrect]</td>
</tr>
<tr>
<td>554</td>
<td>Transaction failed</td>
</tr>
</tbody>
</table>

Summary of UTL_SMTP Subprograms

Table 82–4  UTL_SMTP Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;connection Record Type&quot; on page 82-7</td>
<td>This is a PL/SQL record type used to represent a SMTP connection.</td>
</tr>
<tr>
<td>&quot;reply, replies Record Types&quot; on page 82-8</td>
<td>PL/SQL record types used to represent an SMTP reply line.</td>
</tr>
<tr>
<td>&quot;open_connection Function&quot; on page 82-9</td>
<td>Opens a connection to an SMTP server.</td>
</tr>
<tr>
<td>&quot;command(), command_replies() Functions&quot; on page 82-10</td>
<td>Performs a generic SMTP command.</td>
</tr>
<tr>
<td>&quot;helo Function&quot; on page 82-10</td>
<td>Performs initial handshaking with SMTP server after connecting.</td>
</tr>
<tr>
<td>&quot;ehlo Function&quot; on page 82-11</td>
<td>Performs initial handshaking with SMTP server after connecting, with extended information returned.</td>
</tr>
<tr>
<td>&quot;mail Function&quot; on page 82-12</td>
<td>Initiates a mail transaction with the server. The destination is a mailbox.</td>
</tr>
<tr>
<td>&quot;rcpt Function&quot; on page 82-13</td>
<td>Specifies the recipient of an e-mail message.</td>
</tr>
<tr>
<td>&quot;data Function&quot; on page 82-14</td>
<td>Specifies the body of an e-mail message.</td>
</tr>
<tr>
<td>&quot;open_data(), write_data(), write_raw_data(), close_data() Functions&quot; on page 82-15</td>
<td>Provide more fine-grain control to the data() API.</td>
</tr>
</tbody>
</table>
connection Record Type

This is a PL/SQL record type used to represent an SMTP connection.

Syntax

```plsql
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The name of the remote host when connection is established. NULL when no connection is established.</td>
</tr>
<tr>
<td>port</td>
<td>The port number of the remote SMTP server connected. NULL when no connection is established.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>
reply, replies Record Types

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.

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

```plsql
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>The 3-digit reply code.</td>
</tr>
<tr>
<td>text</td>
<td>The text message of the reply.</td>
</tr>
</tbody>
</table>

Table 82–6 reply, replies Record Type Fields

82-8  Supplied PL/SQL Packages and Types Reference
open_connection Function

This function opens a connection to an SMTP server.

Syntax

```sql
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;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host (IN)</td>
<td>The name of the SMTP server host</td>
</tr>
<tr>
<td>port (IN)</td>
<td>The port number on which SMTP server is listening (usually 25).</td>
</tr>
<tr>
<td>tx_timeout (IN)</td>
<td>The time in seconds that the UTL_SMTP package waits before giving up in a</td>
</tr>
<tr>
<td></td>
<td>read or write operation in this connection. In read operations, this</td>
</tr>
<tr>
<td></td>
<td>package gives up if no data is available for reading immediately. In write</td>
</tr>
<tr>
<td></td>
<td>operations, this package gives up if the output buffer is full and no data</td>
</tr>
<tr>
<td></td>
<td>is to be sent into the network without being blocked. 0 indicates not to</td>
</tr>
<tr>
<td></td>
<td>wait at all. NULL indicates to wait forever.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_SMTP.COMMAND (  
c IN connection,  
cmd IN VARCHAR2,  
arg IN VARCHAR2 DEFAULT NULL)  
RETURN reply;

UTL_SMTP.COMMAND (  
c IN connection,  
arg IN VARCHAR2 DEFAULT NULL);

UTL_SMTP.COMMAND_REPLIES (  
c IN connection,  
cmd IN VARCHAR2,  
arg IN VARCHAR2 DEFAULT NULL)  
RETURN replies;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>cmd (IN)</td>
<td>The SMTP command to send to the server.</td>
</tr>
<tr>
<td>arg (IN)</td>
<td>The optional argument to the SMTP argument. A space will be inserted between <code>cmd</code> and <code>arg</code>.</td>
</tr>
</tbody>
</table>

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;
```

Parameters

**Table 82–9  helo Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>domain (IN NOCOPY)</td>
<td>The domain name of the local (sending) host. Used for identification purposes.</td>
</tr>
</tbody>
</table>

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;
```
mail Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c  (IN NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>domain (IN NOCOPY)</td>
<td>The domain name of the local (sending) host. Used for identification purposes.</td>
</tr>
</tbody>
</table>

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

```sql
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);
```
Summary of UTL_SMTP Subprograms

Parameters

This command does not send the message; it simply begins its preparation. It must be followed by calls to \texttt{rcpt()} and \texttt{data()} to complete the transaction. The connection to the SMTP server must be open and a \texttt{helo()} or \texttt{ehlo()} command must have already been sent.

The expected response from the server is a message beginning with status code 250.

\textbf{rcpt Function}

This function specifies the recipient of an e-mail message.

\textbf{Syntax}

\begin{verbatim}
UTL_SMTP.RCPT (c IN OUT NOCOPY connection, 
  recipient IN OUT NOCOPY, 
  parameters IN OUT NOCOPY) 
RETURN reply; 

c (IN OUT NOCOPY) The SMTP connection.

\end{verbatim}

Table 82–11  Mail Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>sender (IN OUT NOCOPY)</td>
<td>The e-mail address of the user sending the message.</td>
</tr>
<tr>
<td>parameters (IN OUT NOCOPY)</td>
<td>The additional parameters to MAIL command as defined in Section 6 of [RFC1869]. It should follow the format of “XXX=XXX (XXX=XXX ....)”.</td>
</tr>
</tbody>
</table>

Usage Notes

This command does not send the message; it simply begins its preparation. It must be followed by calls to \texttt{rcpt()} and \texttt{data()} to complete the transaction. The connection to the SMTP server must be open and a \texttt{helo()} or \texttt{ehlo()} command must have already been sent.

The expected response from the server is a message beginning with status code 250.

\textbf{rcpt Function}

This function specifies the recipient of an e-mail message.

\textbf{Syntax}

\begin{verbatim}
UTL_SMTP.RCPT (c IN OUT NOCOPY connection, 
  recipient IN OUT NOCOPY, 
  parameters IN OUT NOCOPY) 
RETURN reply; 

c (IN OUT NOCOPY) The SMTP connection.

\end{verbatim}

Table 82–12  \texttt{rcpt} Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP Connection.</td>
</tr>
<tr>
<td>body (IN OUT NOCOPY)</td>
<td>The text of the message to be sent, including headers, in [RFC822] format.</td>
</tr>
</tbody>
</table>
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

```sql
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);
```
open_data(), write_data(), write_raw_data(), close_data() Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roller (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>data (IN OUT NOCOPY)</td>
<td>The portion of the text of the message to be sent, including headers, in [RFC822] format.</td>
</tr>
</tbody>
</table>

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...
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

```sql
UTL_SMTP.RSET (c IN OUT NOCOPY connection) RETURN reply;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_SMTP.VRFY (c IN OUT NOCOPY connection
recipient IN OUT NOCOPY)
RETURN reply;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
<tr>
<td>recipient (IN OUT NOCOPY)</td>
<td>The e-mail address to be verified.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_SMTP.NOOP (c IN OUT NOCOPY connection)
RETURN VARCHAR2;
UTL_SMTP.NOOP (``
Parameter

Table 82–17  noop Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
</tbody>
</table>

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

```sql
UTL_SMTP.QUIT (
    c IN OUT NOCOPY connection)
RETURN VARCHAR2;
```

Parameter

Table 82–18  quit Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The SMTP connection.</td>
</tr>
</tbody>
</table>

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
quit Function

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()
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.

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 from outside of the program.

This chapter discusses the following topics:

- Exceptions
- Example
- Summary of `UTL_TCP` Subprograms
Exceptions

The exceptions raised by the TCP/IP package are listed in Table 83–1.

Table 83–1  TCP/IP Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER_TOO_SMALL</td>
<td>Buffer is too small for input that requires look-ahead.</td>
</tr>
<tr>
<td>END_OF_INPUT</td>
<td>Raised when no more data is available to read from the connection.</td>
</tr>
<tr>
<td>NETWORK_ERROR</td>
<td>Generic network error.</td>
</tr>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument passed in an API call (for example, a negative buffer size).</td>
</tr>
<tr>
<td>TRANSFER_TIMEOUT</td>
<td>No data is read and a read time-out occurred.</td>
</tr>
<tr>
<td>PARTIAL_MULTIBYTE_CHAR</td>
<td>No complete character is read and a partial multi-byte character is found at the end of the input.</td>
</tr>
</tbody>
</table>

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.

```plsql
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 might be used by
an application to send email. The application connects to an SMTP server at port 25
and sends a simple text message.

```plsql
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;connection&quot; on page 83-4</td>
<td>A PL/SQL record type used to represent a TCP/IP connection.</td>
</tr>
<tr>
<td>&quot;CRLF&quot; on page 83-6</td>
<td>The character sequence carriage-return line-feed. It is the newline sequence commonly used many communication standards.</td>
</tr>
<tr>
<td>&quot;open_connection Function&quot; on page 83-6</td>
<td>Opens a TCP/IP connection to a specified service.</td>
</tr>
<tr>
<td>&quot;available Function&quot; on page 83-9</td>
<td>Determines the number of bytes available for reading from a TCP/IP connection.</td>
</tr>
<tr>
<td>&quot;read_raw Function&quot; on page 83-10</td>
<td>Receives binary data from a service on an open connection.</td>
</tr>
<tr>
<td>&quot;write_raw Function&quot; on page 83-11</td>
<td>Transmits a binary message to a service on an open connection.</td>
</tr>
<tr>
<td>&quot;read_text Function&quot; on page 83-12</td>
<td>Receives text data from a service on an open connection.</td>
</tr>
<tr>
<td>&quot;write_text Function&quot; on page 83-14</td>
<td>Transmits a text message to a service on an open connection.</td>
</tr>
<tr>
<td>&quot;read_line Function&quot; on page 83-15</td>
<td>Receives a text line from a service on an open connection.</td>
</tr>
<tr>
<td>&quot;write_line Function&quot; on page 83-16</td>
<td>Transmits a text line to a service on an open connection.</td>
</tr>
<tr>
<td>&quot;get_raw(), get_text(), get_line() Functions&quot; on page 83-17</td>
<td>Convenient forms of the read functions, which return the data read instead of the amount of data read.</td>
</tr>
<tr>
<td>&quot;flush Procedure&quot; on page 83-18</td>
<td>Transmits all data in the output buffer, if a buffer is used, to the server immediately.</td>
</tr>
<tr>
<td>&quot;close_connection Procedure&quot; on page 83-18</td>
<td>Closes an open TCP/IP connection.</td>
</tr>
<tr>
<td>&quot;close_all_connections Procedure&quot; on page 83-19</td>
<td>Closes all open TCP/IP connections.</td>
</tr>
</tbody>
</table>

connection

This is a PL/SQL record type used to represent a TCP/IP connection.
Summary of UTL_TCP Subprograms

Syntax

```plaintext
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_host</td>
<td>The name of the remote host when connection is established. NULL when no connection is established.</td>
</tr>
<tr>
<td>remote_port</td>
<td>The port number of the remote host connected. NULL when no connection is established.</td>
</tr>
<tr>
<td>local_host</td>
<td>The name of the local host used to establish the connection. NULL when no connection is established.</td>
</tr>
<tr>
<td>local_port</td>
<td>The port number of the local host used to establish the connection. NULL when no connection is established.</td>
</tr>
<tr>
<td>charset</td>
<td>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 (i.e. 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.</td>
</tr>
<tr>
<td>newline</td>
<td>The newline character sequence. This newline character sequence is appended to the text line sent by write_line() API.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 83-4  open_connection Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_host (IN)</td>
<td>The name of the host providing the service. When remote_host is NULL, it connects to the local host.</td>
</tr>
<tr>
<td>remote_port (IN)</td>
<td>The port number on which the service is listening for connections.</td>
</tr>
<tr>
<td>local_host (IN)</td>
<td>The name of the host providing the service. NULL means don’t care.</td>
</tr>
<tr>
<td>local_port (IN)</td>
<td>The port number on which the service is listening for connections. NULL means don’t care.</td>
</tr>
<tr>
<td>in_buffer_size (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>out_buffer_size (IN)</td>
<td>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.</td>
</tr>
</tbody>
</table>
open_connection Function

Usage Notes

Note that connections opened by this UTL_TCP package can remain open and be passed from one database call to another in MTS 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.

Table 83–4  open_connection Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset (IN)</td>
<td>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 (i.e. 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.</td>
</tr>
<tr>
<td>newline (IN)</td>
<td>The newline character sequence. This newline character sequence is appended to the text line sent by write_line() API.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```plsql
UTL_TCP.AVAILABLE (    c IN OUT NOCOPY connection,    timeout IN PLS_INTEGER DEFAULT 0) RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to determine the amount of data that is available to be read from.</td>
</tr>
<tr>
<td>timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>

Usage Notes

The connection must have already been opened via 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:

```plsql
DECLARE    c utl_tcp.connection
```
read_raw Function

```plsql
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

```plsql
U TL_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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to receive data from.</td>
</tr>
<tr>
<td>data (IN OUT COPY)</td>
<td>The data received.</td>
</tr>
<tr>
<td>len (IN)</td>
<td>The number of bytes of data to receive.</td>
</tr>
</tbody>
</table>
Summary of UTL_TCP Subprograms

Usage Notes

The connection must have already been opened via 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;
```
read_text Function

**Table 83–7  write_raw Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to send data to.</td>
</tr>
<tr>
<td>data (IN)</td>
<td>The buffer containing the data to be sent.</td>
</tr>
<tr>
<td>len (IN)</td>
<td>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.</td>
</tr>
<tr>
<td>return value</td>
<td>The actual number of bytes of data transmitted.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The connection must have already been opened via 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 83–8  read_text Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to receive data from.</td>
</tr>
<tr>
<td>data (IN OUT NOCOPY)</td>
<td>The data received.</td>
</tr>
<tr>
<td>len (IN)</td>
<td>The number of characters of data to receive.</td>
</tr>
</tbody>
</table>
Usage Notes

The connection must have already been opened via 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 multi-byte, 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 multi-byte character is found at the end of input, this function stops reading and returns all the complete multi-byte 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 multi-byte character can be read as binary by the `read_raw` function. If a partial multi-byte 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.

### Table 83–8 read_text Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>peek (IN)</code></td>
<td>Normally, users want to read the data and remove it from the input queue, i.e. consuming it. In some situations, users may just want to look ahead at the data, i.e. peeking 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 <code>TRUE</code> and an input buffer must be set up when the connection is opened. The amount of data that can be peeked (i.e. read but kept in the input queue) must be less the size of input buffer.</td>
</tr>
<tr>
<td><code>return value</code></td>
<td>The actual number of characters of data received.</td>
</tr>
</tbody>
</table>
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**

```sql
UTL_TCP.WRITE_TEXT (c IN OUT NOCOPY connection,
                      data IN VARCHAR2,
                      len IN PLS_INTEGER DEFAULT NULL)
RETURN PLS_INTEGER;
```

**Table 83–9 write_text Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c</code> (IN OUT NOCOPY)</td>
<td>The TCP connection to send data to.</td>
</tr>
<tr>
<td><code>data</code> (IN)</td>
<td>The buffer containing the data to be sent.</td>
</tr>
<tr>
<td><code>len</code> (IN)</td>
<td>The number of characters of data to transmit. When <code>len</code> is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.</td>
</tr>
<tr>
<td>return value</td>
<td>The actual number of characters of data transmitted.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The connection must have already been opened via 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

```sql
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;
```

Usage Notes

The connection must have already been opened via 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

Table 83–10  read_line Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to receive data from.</td>
</tr>
<tr>
<td>data (IN OUT NOCOPY)</td>
<td>The data received.</td>
</tr>
<tr>
<td>remove_crlf (IN)</td>
<td>If TRUE, the trailing CR/LF character(s) are removed from the received message.</td>
</tr>
<tr>
<td>peek (IN)</td>
<td>Normally, users want to read the data and remove it from the input queue, i.e. consuming it. In some situations, users may just want to look ahead at the data, i.e. peeking 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 an input buffer must be set up when the connection is opened. The amount of data that can be peeked (i.e. read but kept in the input queue) must be less the size of input buffer.</td>
</tr>
<tr>
<td>return value</td>
<td>The actual number of characters of data received.</td>
</tr>
</tbody>
</table>
write_line Function

Successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multi-byte character is found at the end of input, this function stops reading and returns all the complete multi-byte 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 multi-byte character can be read as binary by the read_raw function. If a partial multi-byte 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

```sql
UTL_TCP.WRITE_LINE (c IN OUT NOCOPY connection,
data IN VARCHAR2 DEFAULT NULL)
RETURN PLS_INTEGER;
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to send data to.</td>
</tr>
<tr>
<td>data (IN)</td>
<td>The buffer containing the data to be sent.</td>
</tr>
<tr>
<td>return value</td>
<td>The actual number of characters of data transmitted.</td>
</tr>
</tbody>
</table>

Usage Notes

The connection must have already been opened via 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

\[\text{write_raw()}, \text{write_text()}, \text{flush()}\]

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

\[
\begin{align*}
\text{UTL_TCP.GET\_RAW} & \quad \text{(c IN OUT NOCOPY connection,} \\
& \quad \quad \text{len IN PLS\_INTEGER DEFAULT 1,} \\
& \quad \quad \text{peek IN BOOLEAN DEFAULT FALSE) RETURN RAW;} \\
\text{UTL_TCP.GET\_TEXT} & \quad \text{(c IN OUT NOCOPY connection,} \\
& \quad \quad \text{len IN PLS\_INTEGER DEFAULT 1,} \\
& \quad \quad \text{peek IN BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;} \\
\text{UTL_TCP.GET\_LINE} & \quad \text{(c IN OUT NOCOPY connection,} \\
& \quad \quad \text{remove_crlf IN BOOLEAN DEFAULT false,} \\
& \quad \quad \text{peek IN BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;} \\
\end{align*}
\]

Table 83–12  get\_raw(), get\_text(), and get\_line() Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to receive data from.</td>
</tr>
<tr>
<td>len (IN)</td>
<td>The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.</td>
</tr>
<tr>
<td>peek (IN)</td>
<td>Normally, users want to read the data and remove it from the input queue, i.e. consuming it. In some situations, users may just want to look ahead at the data, i.e. peeking 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 an input buffer must be set up when the connection is opened. The amount of data that can be peeked (i.e. read but kept in the input queue) must be less the size of input buffer.</td>
</tr>
<tr>
<td>remove_crlf (IN)</td>
<td>If TRUE, the trailing CR/LF character(s) are removed from the received message.</td>
</tr>
</tbody>
</table>

Usage Notes

The connection must have already been opened via a call to `open_connection()`. 

UTL\_TCP 83-17
Flush Procedure

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 multi-byte 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

```sql
UTL_TCP.FLUSH (c IN OUT NOCOPY connection);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to send data to.</td>
</tr>
</tbody>
</table>

Usage Notes

The connection must have already been opened via 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

```sql
UTL_TCP.close_CLOSE_CONNECTION (c IN OUT NOCOPY connection);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (IN OUT NOCOPY)</td>
<td>The TCP connection to close.</td>
</tr>
</tbody>
</table>

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()
close_all_connections Procedure
The UTL_URL package has two functions: ESCAPE and UNESCAPE.

See Also: Chapter 78, "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. A URL allows you to easily 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 *unreserved characters*. Any other characters in URLs, including multi-byte 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

In fact, *x-www-form-urlencoded* encoding can be implemented using the UTL_URL.escape function as follows:

```sql
CREATE OR REPLACE FUNCTION form_urlencode (v IN VARCHAR2) AS
  v VARCHAR2;
BEGIN
  v := UTL_URL.escape(v);
  RETURN(v);
END;
```

```sql
```

CREATE OR REPLACE FUNCTION form_urlencode (v IN VARCHAR2) AS
  v VARCHAR2;
BEGIN
  v := UTL_URL.escape(v);
  RETURN(v);
END;
```

```sql```
RETURN VARCHAR2 AS
BEGIN
    RETURN utl_url.escape(replace(v, '"', '+'), TRUE);
END;

**UTL_URL Exceptions**

Table 84–1 lists the exceptions that can be raised when the UTL_URL package API is invoked.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad_url</td>
<td>29262</td>
<td>The URL contains badly formed escape code sequences</td>
</tr>
<tr>
<td>bad_fixed_width_charset</td>
<td>29274</td>
<td>Fixed-width multibyte character set is not allowed as a URL character set</td>
</tr>
</tbody>
</table>

**Summary of UTL_URL Subprograms**

Table 84–2 lists the subprograms that are part of the UTL_URL package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ESCAPE Function&quot; on page 84-3</td>
<td>Returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format</td>
</tr>
<tr>
<td>&quot;UNESCAPE Function&quot; on page 84-5</td>
<td>Unescapes the escape character sequences to their original forms in a URL. Convert the %XX escape character sequences to the original characters</td>
</tr>
</tbody>
</table>

**ESCAPE Function**

This function returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format.

**Syntax**

FUNCTION escape (url IN VARCHAR2,
                  escape_reserved_chars IN BOOLEAN DEFAULT FALSE,
                  url_charset IN VARCHAR2 DEFAULT utl_http.body_charset)
ESCAPE Function

RETURN VARCHAR2;

Parameters

Table 84–3 shows the parameters for the \texttt{ESCAPE} function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{url} \texttt{(IN)}</td>
<td>The original URL</td>
</tr>
<tr>
<td>\texttt{escape_reserved_chars} \texttt{(IN)}</td>
<td>Indicates whether the URL reserved characters should be escaped. If set to \texttt{TRUE}, both the reserved and illegal URL characters are escaped. Otherwise, only the illegal URL characters are escaped. The default value is \texttt{FALSE}.</td>
</tr>
<tr>
<td>\texttt{url_charset} \texttt{(IN)}</td>
<td>When escaping a character (single-byte or multi-byte), what is the target character set that character should be converted to before the character is escaped in \texttt{%hex-code} format? If \texttt{url_charset} is \texttt{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 \texttt{UTL_HTTP} package, whose default value is \texttt{ISO-8859-1}. The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.</td>
</tr>
</tbody>
</table>

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 \texttt{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 \texttt{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
$\logon=\texttt{scott/tiger}$ into the query string of a URL, escape the $ and / separately as \%24\logon=\texttt{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:

\begin{verbatim}
utl_url.escape('http://www.acme.com/a url with space.html')
\end{verbatim}

Returns:

\begin{verbatim}
http://foo.com/a\%20url\%20with\%20space.html
\end{verbatim}

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 \texttt{escape\_reserved\_chars} set to \texttt{TRUE}) and then concatenate it with the rest of the URL. For example:

\begin{verbatim}
url := 'http://www.acme.com/search?check=' || utl_url.escape('Is the use of the "$" sign okay?', \texttt{TRUE});
\end{verbatim}

This expression escapes the question mark (?), dollar sign ($), and space characters in \texttt{‘Is the use of the "$" sign okay?’} but not the ? after \texttt{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 \texttt{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 \texttt{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**

\begin{verbatim}
FUNCTION unescape(
    url IN VARCHAR2,
    url_charset IN VARCHAR2 DEFAULT utl_http.body_charset)
RETURN VARCHAR2;
\end{verbatim}
Parameters

Table 84–3 shows the parameters for the UNESCAPE function.

Table 84–4  UNESCAPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url (IN)</td>
<td>The URL to unescape</td>
</tr>
<tr>
<td>url_charset (IN)</td>
<td>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 &quot;ISO-8859-1&quot;. The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.</td>
</tr>
</tbody>
</table>

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.
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
- Summary of ANYDATA Subprograms
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.

```java
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,
```

Summary of ANYDATA Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BEGINCREATE Static Procedure&quot; on page 85-3</td>
<td>Begins creation process on a new AnyData.</td>
</tr>
<tr>
<td>&quot;PIECEWISE Member Procedure&quot; on page 85-4</td>
<td>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).</td>
</tr>
<tr>
<td>&quot;SET Member Procedures&quot; on page 85-4</td>
<td>Sets the current data value.</td>
</tr>
<tr>
<td>&quot;ENDCREATE Member Procedure&quot; on page 85-6</td>
<td>Ends creation of an AnyData.</td>
</tr>
<tr>
<td>&quot;GETTYPENAME Member Function&quot; on page 85-7</td>
<td>Get the fully qualified type name for the AnyData.</td>
</tr>
<tr>
<td>&quot;GETTYPE Member Function&quot; on page 85-7</td>
<td>Gets the Type of the AnyData.</td>
</tr>
<tr>
<td>&quot;GET Member Functions&quot; on page 85-8</td>
<td>Gets the current data value (which should be of appropriate type).</td>
</tr>
</tbody>
</table>
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).

### BEGINCREATE Static Procedure

This procedure begins the creation process on a new AnyData.

#### Syntax

```plaintext
STATIC PROCEDURE BeginCreate(
    xtype IN OUT NOCOPY AnyType,
    adata OUT NOCOPY AnyData);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtype</td>
<td>The type of the AnyData. (Should correspond to OCI_TYPECODE_OBJECT or a Collection typecode.)</td>
</tr>
<tr>
<td>adata</td>
<td>AnyData being constructed.</td>
</tr>
</tbody>
</table>

#### Exception

DBMS_TYPES.invalid_parameters: xtype is invalid (not fully constructed, etc.)

#### Usage Notes

There is NO NEED to call PieceWise() immediately after this call. Automatically, the construction process begins in a piece-wise manner.
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);

Parameter

Table 85–3  BEGINCREATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The current data value.</td>
</tr>
</tbody>
</table>

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 SetVchar(
    self IN OUT NOCOPY AnyData,
    c IN VARCHAR,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SetVchar2(
    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>"
MEMBER PROCEDURE SetCollection(
    self IN OUT NOCOPY AnyData,
    col IN "<collection_type>",
    last_elem IN boolean DEFAULT FALSE);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An AnyData.</td>
</tr>
<tr>
<td>num</td>
<td>The number, etc., that is to be set.</td>
</tr>
<tr>
<td>last_elem</td>
<td>Relevant only if AnyData represents a collection. Set to TRUE if it is the last element of the collection, FALSE otherwise.</td>
</tr>
</tbody>
</table>

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()
Parameter

**Table 85–5 ENDCREATE Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An AnyData.</td>
</tr>
</tbody>
</table>

**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**

```sql
MEMBER FUNCTION GetTypeName(
    self IN AnyData)
RETURN VARCHAR2;
```

Parameter

**Table 85–6 GETTYPENAME Function Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An AnyData.</td>
</tr>
</tbody>
</table>

**Returns**

Type name of the AnyData.

**GETTYPE Member Function**

This function gets the typecode of the AnyData.

**Syntax**

```sql
MEMBER FUNCTION GetType(
```

```
GET Member Functions

self IN AnyData,
typ OUT NOCOPY AnyType)
RETURN PLS_INTEGER;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An AnyData.</td>
</tr>
<tr>
<td>typ</td>
<td>The AnyType corresponding to the AnyData. May be NULL if it does not represent a user-defined type.</td>
</tr>
</tbody>
</table>

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)
Summary of ANYDATA Subprograms

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An AnyData.</td>
</tr>
</tbody>
</table>
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.
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
- Summary of ANYDATASET Subprograms
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>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>The AnyDataSet needs to be constructed value by value, sequentially.</td>
</tr>
<tr>
<td>Procedure on page 86-3</td>
<td></td>
</tr>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>Creates a new AnyDataSet which can be used to create a set of data values</td>
</tr>
<tr>
<td>Procedure on page 86-3</td>
<td>of the given ANYTYPE.</td>
</tr>
<tr>
<td>ADDINSTANCE Member Procedure</td>
<td>Adds a new data instance to an AnyDataSet.</td>
</tr>
<tr>
<td>Procedure on page 86-4</td>
<td></td>
</tr>
<tr>
<td>PIECEWISE Member Procedure</td>
<td>Sets the MODE of construction/access of the data value to be an attribute</td>
</tr>
<tr>
<td>Procedure on page 86-4</td>
<td>at a time (if the data value is of TYPECODE_OBJECT).</td>
</tr>
<tr>
<td>SET* Member Procedures on page</td>
<td>Sets the current data value.</td>
</tr>
<tr>
<td>86-5</td>
<td></td>
</tr>
<tr>
<td>ENDCREATE Member Procedure</td>
<td>Ends Creation of a AnyDataSet. Other creation functions cannot be called</td>
</tr>
<tr>
<td>Procedure on page 86-7</td>
<td>after this call.</td>
</tr>
</tbody>
</table>
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,
    dtype IN OUT NOCOPY AnyType,
    aset OUT NOCOPY AnyDataSet);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typecode</td>
<td>The typecode for the type of the AnyDataSet.</td>
</tr>
<tr>
<td>dtype</td>
<td>The type of the data values. This parameter is a must for user-defined types like TYPECODE_OBJECT, Collection typecodes, etc.</td>
</tr>
<tr>
<td>aset</td>
<td>The AnyDataSet being constructed.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being constructed.</td>
</tr>
</tbody>
</table>

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 86–4 PIECEWISE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being constructed.</td>
</tr>
</tbody>
</table>

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. 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,
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being accessed.</td>
</tr>
<tr>
<td>num</td>
<td>The number, etc., that is to be set.</td>
</tr>
</tbody>
</table>
**Summary of ANYDATASET Subprograms**

**ANYDATASET TYPE**

**86-7**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_TYPES.invalid_parameters</td>
<td>Invalid parameters (if it is not appropriate to add a number at this point in the creation process).</td>
</tr>
<tr>
<td>DBMS_TYPES.incorrect_usage</td>
<td>Incorrect usage.</td>
</tr>
<tr>
<td>DBMS_TYPES.type_mismatch</td>
<td>When the expected type is different from the passed in type.</td>
</tr>
</tbody>
</table>

**ENDCREATE Member Procedure**

This procedure ends Creation of a AnyDataSet. Other creation functions cannot be called after this call.

**Syntax**

```plaintext
MEMBER PROCEDURE EndCreate(
   self IN OUT NOCOPY AnyDataSet);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being constructed.</td>
</tr>
</tbody>
</table>

**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.
GETTYPE Member Function

Syntax

```sql
MEMBER FUNCTION GetType(
    self     IN AnyDataSet,
    typ     OUT NOCOPY AnyType)
RETURN   PLS_INTEGER;
```

Parameters

Table 86–8  GETTYPE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet.</td>
</tr>
<tr>
<td>typ</td>
<td>The AnyType corresponding to the AnyData. May be NULL if it does not represent a user-defined function.</td>
</tr>
</tbody>
</table>

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

```plaintext
MEMBER FUNCTIONGetInstance(
    self IN OUT NOCOPY AnyDataSet)
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being accessed.</td>
</tr>
</tbody>
</table>

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**

```plsql
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>')
### Parameters

**Table 86–10 GET* Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being accessed.</td>
</tr>
<tr>
<td>num</td>
<td>The number, etc., that is to be obtained.</td>
</tr>
</tbody>
</table>

### 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

```plsql
MEMBER FUNCTION GetCount(
    self IN AnyDataSet)
```
GETCOUNT Member Function

```
RETURN PLS_INTEGER;
```

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyDataSet being accessed.</td>
</tr>
</tbody>
</table>

### Returns

The number of data instances.
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.

This chapter discusses the following:

- Summary of ANYTYPE Subprograms
Summary of ANYTYPE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>Creates a new instance of ANYTYPE which can be used to create a transient type description.</td>
</tr>
<tr>
<td>SETINFO Member Procedure</td>
<td>Sets any additional information required for constructing a COLLECTION or builtin type.</td>
</tr>
<tr>
<td>ADDATTR Member Procedure</td>
<td>Adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPETCODE_OBJECT).</td>
</tr>
<tr>
<td>ENDCREATE Member Procedure</td>
<td>Ends creation of a transient AnyType. Other creation functions cannot be called after this call.</td>
</tr>
<tr>
<td>GETPERSISTENT Static Function</td>
<td>Returns an AnyType corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.</td>
</tr>
<tr>
<td>GETINFO Member Function</td>
<td>Gets the type information for the AnyType.</td>
</tr>
<tr>
<td>GETATTRELEMINFO Member Function</td>
<td>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.</td>
</tr>
</tbody>
</table>

BEGINCREATE Static Procedure

This procedure creates a new instance of ANYTYPE which can be used to create a transient type description.

Syntax

```plaintext
STATIC PROCEDURE BEGINCREATE(
    typecode IN PLS_INTEGER,
    atype OUT NOCOPY AnyType);
```
Parameters

**Table 87–2** BEGINCREATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typecode</td>
<td>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.</td>
</tr>
<tr>
<td>atype</td>
<td>AnyType for a transient type</td>
</tr>
</tbody>
</table>

**SETINFO Member Procedure**

This procedure sets any additional information required for constructing a COLLECTION or builtin type.

**Syntax**

```plaintext
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 87–3** SETINFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient ANYTYPE that is being constructed.</td>
</tr>
<tr>
<td>prec, scale (OPTIONAL)</td>
<td>Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
</tbody>
</table>
ADDATTR Member Procedure

Table 87–3  SETINFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len (OPTIONAL)</td>
<td>Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid, csfrm (OPTIONAL)</td>
<td>Required if typecode represents types requiring character information such as CHAR, VARCHAR, VARCHAR2, or CFILE.</td>
</tr>
<tr>
<td>atype (OPTIONAL)</td>
<td>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.</td>
</tr>
</tbody>
</table>

The Following Parameters Are Required For Collection Types:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem_tc</td>
<td>Must be of the collection element’s typecode (from DBMS_TYPES package).</td>
</tr>
<tr>
<td>elem_count</td>
<td>Pass 0 for elem_count if the self represents a nested table (TYPECODE_TABLE). Otherwise pass the collection count if self represents a VARRAY.</td>
</tr>
</tbody>
</table>

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,
  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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem_tc</td>
<td>Must be of the collection element’s typecode (from DBMS_TYPES package).</td>
</tr>
<tr>
<td>elem_count</td>
<td>Pass 0 for elem_count if the self represents a nested table (TYPECODE_TABLE). Otherwise pass the collection count if self represents a VARRAY.</td>
</tr>
</tbody>
</table>

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,
```
Summary of ANYTYPE Subprograms

ANYTYPE TYPE

87-5

scale IN PLS_INTEGER,
len IN PLS_INTEGER,
csid IN PLS_INTEGER,
csf rm IN PLS_INTEGER,
attr_type IN ANYTYPE DEFAULT NULL);

Parameters

Table 87–4  ADDATTR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient AnyType that is being constructed. Must be of type DBMS_TYPES.TYPECODE_OBJECT.</td>
</tr>
<tr>
<td>aname (OPTIONAL)</td>
<td>Attribute’s name. Could be NULL.</td>
</tr>
<tr>
<td>typecode</td>
<td>Attribute’s typecode. Can be built-in or user-defined typecode (from DBMS_TYPES package).</td>
</tr>
<tr>
<td>prec, scale (OPTIONAL)</td>
<td>Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len (OPTIONAL)</td>
<td>Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Give length.</td>
</tr>
<tr>
<td>csid, csfrm (OPTIONAL)</td>
<td>Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, VARCHAR2, CFILE.</td>
</tr>
<tr>
<td>attr_type (OPTIONAL)</td>
<td>AnyType corresponding to a user-defined type. This parameter is required if the attribute is a user defined type.</td>
</tr>
</tbody>
</table>

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);

ANYTYPE TYPE 87-5
GETPERSISTENT Static Function

This procedure returns an AnyType corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

Syntax

```sql
STATIC FUNCTION GetPersistent(
    schema_name IN VARCHAR2,
    type_name IN VARCHAR2,
    version IN VARCHAR2 DEFAULT NULL
) RETURN AnyType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient AnyType that is being constructed.</td>
</tr>
</tbody>
</table>

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

```sql
MEMBER FUNCTION GetInfo ( 
    self IN AnyType,
```
Summary of ANYTYPE Subprograms

ANYTYPE TYPE

87-7

prec OUT PLS_INTEGER,
scale OUT PLS_INTEGER,
len OUT PLS_INTEGER,
csid OUT PLS_INTEGER,
csfm OUT PLS_INTEGER,
schema_name OUT VARCHAR2,
type_name OUT VARCHAR2,
version OUT varchar2,
count OUT PLS_INTEGER)
RETURN PLS_INTEGER;

Parameters

Table 87–7  GETINFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyType.</td>
</tr>
<tr>
<td>prec, scale</td>
<td>If typecode represents a number. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid, csfrm</td>
<td>If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE.</td>
</tr>
<tr>
<td>schema_name, type_name, version</td>
<td>Type’s schema (if persistent), typename and version.</td>
</tr>
<tr>
<td>count</td>
<td>If self is a VARRAY, this gives the VARRAY count. If self is of TYPECODE_OBJECT, this gives the number of attributes.</td>
</tr>
</tbody>
</table>

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

```plsql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The AnyType.</td>
</tr>
<tr>
<td>pos</td>
<td>If self is of TYPECODE_OBJECT, this gives the attribute position (starting at 1). It is ignored otherwise.</td>
</tr>
<tr>
<td>prec, scale</td>
<td>If attribute/collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid, csfrm</td>
<td>If typecode represents a type requiring character information such as: CHAR, VARCHAR, VARCHAR2, CFILE. Gives character set ID, character set form.</td>
</tr>
<tr>
<td>attr_elt_type</td>
<td>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.</td>
</tr>
<tr>
<td>aname</td>
<td>Attribute name (if it is an attribute of an object type, NULL otherwise).</td>
</tr>
</tbody>
</table>
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).
GETATTRELEMINFO Member Function
A

Advanced Queuing
  DBMS_AQADM package, 5-1
altering
  propagation method, 48-28, 48-32
  savepoints, 70-5
  workspaces, 70-6
AlterSavepoint procedure, 70-5
AlterWorkspace procedure, 70-6
anonymous PL/SQL blocks
  dynamic SQL and, 62-3
arrays
  BIND_ARRAY procedure, 62-7
  bulk DML using DBMS_SQL, 62-29
availability
  extended, 48-11, 48-31, 48-87, 48-101, 48-107, 48-111
available() function of UTL_TCP, 83-9

B

BeginResolve procedure, 70-7

C

catproc.sql script, 1-3
Change Data Capture
  DBMS_LOGMNR_CDC_PUBLISH package, 25-1
  DBMS_LOGMNR_CDC_SUBSCRIBE package, 26-1
character sets
  ANY_CS, 22-3
child workspace
  merging, 70-40, 70-43
  refreshing, 70-41, 70-43
  removing, 70-45
CLOB datatype
  NCLOBs, 22-3
CLOBs (character large objects) datatype
  NCLOBS, 22-3
close_all_connections() function of UTL_TCP, 83-19
close_connection() function of UTL_TCP, 83-18
close_data() function of UTL_SMTP, 82-15
collections
  table items, 62-29
columns
  adding to master tables, 48-98
column groups
  adding members to, 48-8
  creating, 48-61, 48-85
dropping, 48-66
  removing members from, 48-67
command() function of UTL_SMTP, 82-10
command_replies() function of UTL_SMTP, 82-10
CommitResolve procedure, 70-8
comparing
  tables, 44-3
compressing workspaces, 70-9, 70-11
CompressWorkspace procedure, 70-9
CompressWorkspaceTree procedure, 70-11
conflict management, 70-46
beginning resolution, 70-7
committing resolution, 70-8
rolling back resolution, 70-51
showing conflicts, 70-56
conflict resolution
additive method, 48-20
statistics, 48-40, 48-94
connection
function of UTL_TCP, 83-4
connection function
of UTL_SMTP, 82-7
context of current operation
getting, 70-26
continually refreshed workspace, 70-16
CopyForUpdate procedure, 70-13
CREATE PACKAGE BODY command, 1-3
CREATE PACKAGE command, 1-3
CreateSavepoint procedure, 70-14
CreateWorkspace procedure, 70-16
creating
new workspaces, 70-16
packages, 1-3
savepoints, 70-14
CRLF (carriage-return line-feed)
function of UTL_TCP, 83-6
cursors
DBMS_SQL package, 62-5

D

data definition language
altering replicated objects, 48-29
asynchronous, 48-80
supplying asynchronous, 48-80
data() function
of UTL_SMTP, 82-14
database tables
creating for DBMS_TRACE, 64-3
datatypes
DBMS_DESCRIBE, 13-4
DESC_TAB, 62-44
PL/SQL
numeric codes for, 13-8
ROWID, 57-1
DBA_REPCATLOG view
purging, 48-88
DBA_REPCOLUMN_GROUP view
updating, 48-41
DBA_REPGROUP view
updating, 48-44
DBA_REPOBJECT view
updating, 48-45
DBA_REPPRIORITY_GROUP view
updating, 48-43
DBA_REPRESOLUTION view
updating, 48-48
DBA_REPRESOLUTION_STATISTICS view
purging, 48-89
DBA_REPSITES view
updating, 48-46
DBMS_ALERT package, 2-1
DBMS_APPLICATION_INFO package, 3-2
DBMS_AQ package, 4-1
DBMS_AQADM package, 5-1
DBMS_AQELM package, 6-1, 6-2
DBMS_BACKUP_RESTORE package, 7-1
DBMS_DDL package, 8-1
DBMS_DEBUG package, 9-1
DBMS_DEFER package, 10-1
ANY_CHAR_ARG procedure, 10-5
ANY_CLOB_ARG procedure, 10-5
ANY_VARCHAR2_ARG procedure, 10-5
ANYDATA_ARG procedure, 10-5
BLOB_ARG procedure, 10-5
CALL procedure, 10-3
CHAR_ARG procedure, 10-5
CLOB_ARG procedure, 10-5
COMMIT_WORK procedure, 10-4
datatype_ARG procedure, 10-5
DATE_ARG procedure, 10-5
IDS_ARG procedure, 10-5
IYM_ARG procedure, 10-5
NCHAR_ARG procedure, 10-5
NCLOB_ARG procedure, 10-5
NUMBER_ARG procedure, 10-5
NVARCHAR2_ARG procedure, 10-5
RAW_ARG procedure, 10-5
ROWID_ARG procedure, 10-5
TIMESTAMP_ARG procedure, 10-5
TRANSACTION procedure, 10-8
TSLTZ_ARG procedure, 10-5
TSTZ_ARG procedure, 10-5
VARCHAR2_ARG procedure, 10-5
DBMS_DEFER_QUERY package, 11-1
GET_ANYDATA_ARG procedure, 11-9
GET_ARG_FORM function, 11-3
GET_ARG_TYPE function, 11-5
GET_BLOB_ARG procedure, 11-9
GET_CALL_ARGS procedure, 11-7
GET_CHAR_ARG procedure, 11-9
GET_CLOB_ARG procedure, 11-9
GET_datatype_ARG function, 11-9
GET_DATE_ARG procedure, 11-9
GET_IDS_ARG procedure, 11-9
GET_IYM_ARG procedure, 11-9
GET_NCHAR_ARG procedure, 11-9
GET_NCLOB_ARG procedure, 11-9
GET_NUMBER_ARG procedure, 11-9
GET_OBJECT_NULL_VECTOR_ARG function, 11-12
GET_RAW_ARG procedure, 11-9
GET_ROWID_ARG procedure, 11-9
GET_TIMESTAMP_ARG procedure, 11-9
GET_TSLTZ_ARG procedure, 11-9
GET_TSTZ_ARG procedure, 11-9
GET_VARCHAR2_ARG procedure, 11-9
SET_DISABLED procedure, 12-24
UNREGISTER_PROPAGATOR procedure, 12-26
UNSCHEDULE_PURGE procedure, 12-27
UNSCHEDULE_PUSH procedure, 12-27
DBMS_DESCRIBE package, 13-1
DBMS_DISTRIBUTED_TRUST_ADMIN package, 14-1
DBMS_FGA package, 15-1
DBMS_FLASHBACK package, 16-1, 16-6
DBMS_HS_PASSTHROUGH package, 17-1
DBMS_IOT package, 18-1
DBMS_JOB package, 19-1
and instance affinity, 19-2
DBMS_LOB package, 22-1
DBMS_LOCK package, 23-1
DBMS_LOGMNR package, 24-1
ADD_LOGFILE procedure, 24-4
COLUMN_PRESENT function, 24-9
constants, 24-2
END_LOGMNR procedure, 24-8
MINE_VALUE function, 24-8
START_LOGMNR procedure, 24-5
DBMS_LOGMNR_CDC_PUBLISH package, 25-1
ALTER_CHANGE_TABLE procedure, 25-8
CREATE_CHANGE_SOURCE procedure, 25-3
CREATE_CHANGE_TABLE procedure, 25-3
DROP_CHANGE_TABLE procedure, 25-16
DBMS_LOGMNR_CDC_SUBSCRIBE package, 26-1
ADD_DEFAULT_DEST procedure, 12-4
CLEAR_PROP_STATISTICS procedure, 12-5
DELETE_DEF_DESTINATION procedure, 12-6
DELETE_DEFAULT_DEST procedure, 12-6
DELETE_ERROR procedure, 12-7
DELETE_TRAN procedure, 12-8, 12-9, 12-11
DISABLE function, 12-9
EXCLUDE_PUSH function, 12-10
EXECUTE_ERROR procedure, 12-11
EXECUTE_ERROR_AS_USER procedure, 12-12
PURGE function, 12-13
PUSH function, 12-16
REGISTER_PROPAGATOR procedure, 12-19
SCHEDULE_EXECUTION procedure, 12-22
SCHEDULE_PURGE procedure, 12-20
SCHEDULE_PUSH procedure, 12-22
PREPARE_SUBSCRIBER_VIEW procedure, 26-11
PREPARE_UNBOUNDED_VIEW procedure, 26-13
PURGE_WINDOW procedure, 26-14
SUBSCRIBE procedure, 26-6
usage examples, 26-16
DBMS_LOGMNR_D package, 27-1
REGISTER_USER_REPGROUP procedure, 49-5
REVOKE_ADMIN_ANY_SCHEMA procedure, 49-7
REVOKE_ADMIN_SCHEMA procedure, 49-8
UNREGISTER_USER_REPGROUP procedure, 49-9
DBMS_REPCAT_INSTANTIATE package
DROP_SITE_INSTANTIATION procedure, 50-3
INSTANTIATE_OFFLINE function, 50-3
INSTANTIATE_ONLINE function, 50-6
DBMS_REPCAT_RGT package
ALTER_REFRESH_TEMPLATE procedure, 51-5
ALTER_TEMPLATE_OBJECT procedure, 51-7
ALTER_USER_AUTHORIZATION procedure, 51-12
ALTER_USER_PARM_VALUE procedure, 51-14
COMPARE_TEMPLATES function, 51-16
COPY TEMPLATE function, 51-18
CREATE OBJECT_FROM_EXISTING function, 51-20
CREATE_REFRESH TEMPLATE function, 51-22
CREATE_TEMPLATE_OBJECT function, 51-24
CREATE_TEMPLATE_PARM function, 51-27
CREATE_USER_AUTHORIZATION function, 51-30
CREATE_USER_PARM_VALUE function, 51-31
DELETE_RUNTIME_PARMS procedure, 51-33
DROP_ALL_OBJECTS procedure, 51-34
DROP_ALL_TEMPLATE_PARM procedure, 51-36
DROP_ALL_TEMPLATE_SITES procedure, 51-37
DROP_ALL_TEMPLATES procedure, 51-38
DROP_ALL_USER_AUTHORIZATIONS procedure, 51-38
DROP_ALL_USER_PARM_VALUES procedure, 51-39
DROP_REFRESH TEMPLATE procedure, 51-40
DROP_SITE_INSTANTIATION procedure, 51-41
DROP_TEMPLATE_OBJECT procedure, 51-42
DROP_TEMPLATE_PARM procedure, 51-44
DROP_USER_AUTHORIZATION procedure, 51-45
DROP_USER_PARM_VALUE procedure, 51-46
GET_RUNTIME_PARM_ID function, 51-47
INSERT_RUNTIME_PARMS procedure, 51-47
INSTANTIATE_OFFLINE function, 51-49
INSTANTIATE_ONLINE function, 51-52
LOCK TEMPLATE EXCLUSIVE procedure, 51-55
LOCK TEMPLATE_SHARED procedure, 51-55
DBMS_REPUTIL package
FROM_REMOTE function, 52-4
GLOBAL_NAME function, 52-5
MAKE_INTERNAL_PKG procedure, 52-5
REPLICATION IS ON function, 52-4
REPLICATION OFF procedure, 52-3
REPLICATION ON procedure, 52-3
SYNC_UP_REP procedure, 52-6
DBMS_RESOURCE_MANAGER package, 53-1
DBMS_RESOURCE_MANAGER_PRIVS package, 54-1
DBMS_RESUMABLE package, 55-1
DBMS_RLS package, 56-1
DBMS_ROWID package, 57-1
DBMS_SESSION package, 58-1
DBMS_SHARED_POOL package, 59-1
DBMS_SPACE package, 60-1
DBMS_SPACE_ADMIN package, 61-1
DBMS_STATS package, 63-1
DBMS_TRACE package, 64-1
DBMS_TRANSACTION package, 65-1
DBMS_TRANSFORM package, 66-1
DBMS_TTS package, 67-1
DBMSUTILITY package, 69-1
DBMS_WM package, 70-1
DDL. See data definition language
DEBUG_EXPTOC package, 74-1
DEFDEFAULTDEST view adding destinations to, 12-4
removing destinations from, 12-6
defered transactions
DefDefaultDest table
  removing destinations from, 12-6
DEFDEFAULTDEST view
  adding destination to, 12-4
  removing destinations from, 12-6
defered remote procedure calls (RPCs)
  argument types, 11-5
  argument values, 11-9
  arguments to, 10-5
  building, 10-3
  executing immediately, 12-16
DEFSCHEDULE view
  clearing statistics, 12-5
  deleting from queue, 12-8
  re-executing, 12-11
  scheduling execution, 12-22
  starting, 10-8
DEFERROR view
  deleting transactions from, 12-7
DEFSCHEDULE view
  clearing statistics, 12-5
DeleteSavepoint procedure, 70-17
deleting
  savepoints, 70-17
  workspaces, 70-44
deployment templates
  alter object, 51-7
  alter parameters, 51-10
  alter template, 51-5
  alter user authorization, 51-12
  alter user parameter values, 51-14
  compare templates, 51-16
  copy template, 51-18
  create object from existing, 51-20
  create template, 51-22
  drop site instantiation, 50-3
  dropping, 51-40
  dropping all, 51-38
  lock template, 51-55
  objects
    creating, 51-24
    dropping, 51-42
    dropping all, 51-34
  offline instantiation, 50-3, 51-49
  online instantiation, 50-6, 51-52
parameters
  creating, 51-27
  dropping, 51-44
  dropping all, 51-36
runtime parameters
  creating, 51-47
  deleting, 51-33
  get ID, 51-47
  inserting, 51-47
sites
  dropping, 51-41
  dropping all, 51-37
user authorizations
  creating, 51-30
  dropping, 51-45
  dropping all, 51-38
user parameter values
  creating, 51-31
  dropping, 51-46
  dropping all, 51-39
DESC_TAB datatype, 62-44
DESDecrypt procedure, 30-6, 30-11
DESEncrypt procedure, 30-5, 30-9
differences
  between tables, 44-3
  rectifying, 44-6
DisableVersioning procedure, 70-19
disabling
  propagation, 12-24
  disabling changes, 70-22
  unfreezing, 70-66
dynamic SQL
  anonymous blocks and, 62-3
  DBMS_SQL functions, using, 62-3
  execution flow in, 62-5
E
ehlo() function
  of UTL_SMTP, 82-11
e-mail
  sending with UTL_SMTP, 82-3
EnableVersioning procedure, 70-21
errors
  returned by DBMS_ALERT package, 18-3
exclusive locks, 70-60
execution flow
    in dynamic SQL, 62-5
extend window
    to create a new view, 26-2
extended availability, 48-11, 48-31, 48-87, 48-101, 48-107, 48-111

F
features, new, xxvii
filenames
    normalization of with DBMS_BACKUP_RESTORE, 7-2
fine-grained access control
    DBMS_RLS package, 56-1
flush()
    function of UTL_TCP, 83-18
FORCE parameter
    and job-to-instance affinity, 19-2
FreezeWorkspace procedure, 70-22
freezing workspace changes, 70-22
    unfreezing, 70-66
functions
    GetConflictWorkspace, 70-24
    GetDiffVersions, 70-24
    GetLockMode, 70-25
    GetMultiWorkspaces, 70-26
    GetOpContext, 70-26
    GetPrivs, 70-27
    GetWorkspace, 70-28
    IsWorkspaceOccupied, 70-36
See also procedures, 70-28

G
get_host_address()
    function of UTL_INADDR, 79-3
get_line()
    function UTL_TCP, 83-17
get_raw()
    function of UTL_TCP, 83-17
get_text()
    function of UTL_TCP, 83-17
GetConflictWorkspace function, 70-24
GetDiffVersions function, 70-24
GetLockMode function, 70-25
GetMultiWorkspaces, 70-26
GetOpContext function, 70-26
GetPrivs function, 70-27
GetWorkspace function, 70-28
GotoWorkspace procedure, 70-31
granting privileges
    system, 70-32
    workspace, 70-34
GrantSystemPriv procedure, 70-32
GrantWorkspacePriv procedure, 70-34

H
helo() function
    of UTL_SMTP, 82-10
hierarchy
    removing, 70-45

I
importing
    materialized views
        offline instantiation and, 33-3, 33-5
        replication groups
            offline instantiation and, 32-4, 32-7
        status check, 48-99
instantiation
    DROP_SITE_INSTANTIATION
        procedure, 50-3, 51-41
    offline
        INSTANTIATE_OFFLINE function, 50-3, 51-49
    online
        INSTANTIATE_ONLINE function, 50-6, 51-52
internet addressing
    using UTL_INADDR, 79-1
IsWorkspaceOccupied function, 70-36

J
jobs
    queues for
removing jobs from, 12-27

**L**

LOB columns with versioned tables, 70-13

LOBs

- DBMS_LOB package, 22-1

lock mode

- getting, 70-25

locking table rows, 70-37

LockRows procedure, 70-37

locks

- disabling, 70-59
- enabling, 70-60

**M**

mail() function

- of UTL_SMTP, 82-12

master definition sites

- relocating, 48-95

master groups

- creating, 48-52
- dropping, 48-68
- quiescing, 48-109
- resuming replication activity, 48-100

master sites

- creating, 48-9
- dropping, 48-97
- propagating changes between, 12-22

master tables

- adding columns to, 48-98

materialized view groups

- creating, 48-57

materialized view logs

- master table purging, 29-8, 29-9, 29-10

materialized view sites

- changing masters, 48-110
- dropping, 48-71
- propagating changes to master, 12-22
- refreshing, 48-90

materialized views

- generating support for, 48-82
- offline instantiation of, 33-3, 33-5
- refreshing, 29-12, 29-15, 29-16

MergeTable procedure, 70-38, 70-43

MergeWorkspace procedure, 70-40

merging table changes, 70-38

merging tables, 70-43

merging workspaces, 70-40

migration

- post-migration actions, 7-1, 30-1

**N**

national language support

- NCLOBs, 22-3

new features, xxvii

noop() function

- of UTL_SMTP, 82-18

**O**

objects

- adding to materialized view sites, 48-58
- altering, 48-29
- creating, 48-53
  - for master group, 48-52
  - for master sites, 48-53
  - for materialized view sites, 48-58
- dropping
  - materialized view site, 48-72
  - generating replication support for, 48-83

offline instantiation

- INSTANTIATE_OFFLINE function, 50-3, 51-49
- materialized views, 33-3, 33-5
- replication groups, 32-3, 32-4, 32-6, 32-7, 32-9

online instantiation

- INSTANTIATE_ONLINE function, 50-6, 51-52

open_connection() function of UTL_TCP, 83-6

open_connection() function of UTL_SMTP, 82-9

open_data() function of UTL_SMTP, 82-15

operation context

- getting, 70-26

OR REPLACE clause

- for creating packages, 1-3
Oracle Advanced Queuing (Oracle AQ)
DBMS_AQADM package, 5-1
OUTLN_PKG package, 37-1

P
package overview, 1-2
package variables
  i_am_a_refresh, 29-7
packages
  creating, 1-3
  referencing, 1-6
  where documented, 1-7
parent workspace
  conflicts with, 70-56
plan stability, 37-1
PL/SQL
datatypes, 13-6
  numeric codes for, 13-8
priority groups
  adding members to, 48-17
  altering members
    priorities, 48-34
    values, 48-35
  creating, 48-62
  dropping, 48-74
  removing members from, 48-73, 48-75
site priority groups
  adding members to, 48-19
privileges
  getting, 70-27
  granting, 70-32, 70-34
  revoking, 70-48, 70-50
procedures
  AlterSavepoint, 70-5
  AlterWorkspace, 70-6
  BeginResolve, 70-7
  CommitResolve, 70-8
  CompressWorkspace, 70-9
  CompressWorkspaceTree, 70-11
  CopyForUpdate, 70-13
  CreateSavepoint, 70-14
  CreateWorkspace, 70-16
  DeleteSavepoint, 70-17
  DisableVersioning, 70-19
  EnableVersioning, 70-21
  FreezeWorkspace, 70-22
  GotoWorkspace, 70-31
  GrantSystemPriv, 70-32
  GrantWorkspacePriv, 70-34
  LockRows, 70-37
  MergeTable, 70-38, 70-43
  MergeWorkspace, 70-40
  RefreshTable, 70-41
  RefreshWorkspace, 70-43
  RemoveWorkspace, 70-44
  RemoveWorkspaceTree, 70-45
  ResolveConflicts, 70-46
  RevokeSystemPriv, 70-48
  RevokeWorkspacePriv, 70-50
  RollbackResolve, 70-51
  RollbackTable, 70-52
  RollbackToSP, 70-53
  RollbackWorkspace, 70-55
  SetConflictWorkspace, 70-56
  SetDiffVersions, 70-57
  SetLockingOFF, 70-59
  SetLockingON, 70-60
  SetMultiWorkspaces, 70-61
  SetWoOverwriteOFF, 70-62
  SetWoOverwriteON, 70-63
  SetWorkspaceLockModeOFF, 70-64
  SetWorkspaceLockModeON, 70-65
  UnfreezeWorkspace, 70-66
  UnlockRows, 70-67
programmatic environments, 4-13
propagation
  altering method, 48-28, 48-32
  disabling, 12-24
  of changes, 48-28
  status of, 12-9
propagator
  registering, 12-19
purging
  DBA_REPCATLOG table, 48-88

Q
queuing
  DBMS_AQADM package, 5-1
exquising
  master groups, 48-109
quit() function
  of UTL_SMTP, 82-19
R
rcpt() function
  of UTL_SMTP, 82-13
read_line()
  function of UTL_TCP, 83-15
read_raw()
  function of UTL_TCP, 83-10
read_text()
  function of UTL_TCP, 83-12
rectifying
tables, 44-6
refresh
  materialized view sites, 48-90
  materialized views, 29-12, 29-15, 29-16
refresh groups
  adding members to, 46-3
  creating, 46-7
  deleting, 46-6
  refresh interval
    changing, 46-4
    refreshing
      manually, 46-10
      removing members from, 46-10
refreshing tables, 70-41
refreshing workspaces, 70-43
RefreshTable procedure, 70-41
RefreshWorkspace procedure, 70-43
registering
  propagator for local database, 12-19
RemoveWorkspace procedure, 70-44
RemoveWorkspaceTree procedure, 70-45
removing workspaces, 70-44
replicated objects
  dropping from master sites, 48-69
replication
datetime datatypes
  abbreviations, 1-7
disabling, 52-3
enabling, 52-3
interval datatypes
  abbreviations, 1-7
replication groups
  offline instantiation of, 32-3, 32-4, 32-6, 32-7, 32-9
replies function
  of UTL_SMTP, 82-8
reply functions
  of UTL_SMTP, 82-8
ResolveConflicts procedure, 70-46
resolving conflicts, 70-46
  beginning, 70-7
  committing, 70-8
  rolling back, 70-51
resuming replication activity, 48-100
RevokeSystemPriv procedure, 70-48
RevokeWorkspacePriv procedure, 70-50
revoking privileges, 70-48, 70-50
RollbackResolve procedure, 70-51
RollbackTable procedure, 70-52
RollbackToSP procedure, 70-53
RollbackWorkspace procedure, 70-55
rolling back
tables, 70-52
  workspaces, 70-55
  to savepoint, 70-53
ROWID datatype
  DBMS_ROWID package, 57-1
  extended format, 57-12
rows
  locking, 70-37
  unlocking, 70-67
rset() function
  of UTL_SMTP, 82-17
S
savepoints
  altering, 70-5
  creating, 70-14
  deleting, 70-17
  rolling back to, 70-53
SDO_CD package, 1-15
SDO_GEOM package, 1-15
SDO_LRS package, 1-16
SDO_MIGRATE package, 1-19
SDO_TUNE package, 1-20
set_disabled, 12-24
SetConflictWorkspace procedure, 70-56
SetDiffVersions procedure, 70-57
SetLockingON procedure, 70-59, 70-60
SetMultiWorkspaces procedure, 70-61
SetWoOverwriteOFF procedure, 70-62
SetWoOverwriteON procedure, 70-63
SetWorkspaceLockModeOFF procedure, 70-64
SetWorkspaceLockModeON procedure, 70-65
shared locks, 70-60
site priority
altering, 48-37
site priority groups
adding members to, 48-19
creating
syntax, 48-64
dropping, 48-76
removing members from, 48-77
SQL statements
larger than 32 KB, 62-26
SQL*Plus
creating a sequence, 1-6
statistics
clearing, 12-5
collecting, 48-94
purging, 48-89
status
propagation, 12-9
stored outlines
OUTLN_PKG package, 37-1
subscriber view
dropping, 26-2
subscriber views
removing, 26-2
subscribers
drop the subscriber view, 26-2
drop the subscription, 26-2
extend the window to create a new view, 26-2
purge the subscription window, 26-2
removing subscriber views, 26-2
retrieve change data from the subscriber views, 26-2
subscription window
p purging, 26-2
SYS_ANYDATA, 11-9
system privileges, 70-32
T
tables
comparing, 44-3
rectifying, 44-6
table items as arrays, 62-29
TRACETAB.SQL, 64-3
U
UnfreezeWorkspace procedure, 70-66
unlocking table rows, 70-67
UnlockRows procedure, 70-67
upgrading
post-upgrade actions, 7-1, 30-1
UTL_COLL package, 75-1
UTL_ENCODE package, 76-1
UTL_FILE package, 77-1
UTL_INET package, 79-1
UTL_PG package, 1-20
UTL_RAW package, 76-1, 80-1
UTL_REF package, 81-1
UTL_SMTP package, 82-1
UTL_TCP package, 83-1
V
versioning
disabling, 70-19
enabling, 70-21
VIEW_WO_OVERWRITE mode
disabling, 70-62
enabling, 70-63
vrfy() function
of UTL_SMTP, 82-18
W
workspace
continually refreshed, 70-16
creating, 70-16
freezing, 70-22
getting, 70-28
selecting, 70-31
unfreezing, 70-66
workspace lock mode
disabling, 70-64
enabling, 70-65
write_data() function
of UTL_SMTP, 82-15
write_line()
function of UTL_TCP, 83-16
write_raw()
function of UTL_TCP, 83-11
write_raw_data() function
of UTL_SMTP, 82-15
write_text()
function of UTL_TCP, 83-14