Oracle8*i*

Replication API Reference

Release 8.1.5

February 1999

A67793-01

ORACLE®

Oracle8i Replication API Reference, Release 8.1.5

A67793-01

Primary Author: William Creekbaum

Contributors: Alan Downing, Harry Sun, Al Demers, Maria Pratt, Jim Stamos, Curt Elsbernd, Pat McElroy, Denis Goddard, and others

Graphic Designer: Valarie Moore

Contents

3 Create a Master Group

4 Create Deployment Template

5 Create a Snapshot Group

6 Conflict Resolution

7 Manage Replicated Environment with APIs

8 Replication Management API Reference

Send Us Your Comments

Oracle8i Replication API Reference, Release 8.1.5

A67793-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

- Electronic mail - infodev@us.oracle.com
- FAX - (650) 506-7228 Attn: Oracle Server Documentation
- Postal service:
  Oracle Corporation
  Server Documentation Manager
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This Preface contains the following topics:

- Overview of the Oracle8i Replication API Reference Book.
- Audience.
- How The Oracle8i Replication API Reference Book Is Organized.
- Conventions Used in This Manual.
- Your Comments Are Welcome.

The *Oracle8i Replication API Reference* contains information relating to both Oracle8*i* and the Oracle8*i* Enterprise Edition. Some features documented in this manual are available only with the Oracle8*i* Enterprise Edition. Furthermore, some of these features are only available if you have purchased a particular option, such as the Objects Option.

For information about the differences between Oracle8*i* and the Oracle8*i* Enterprise Edition, please refer to *Getting to Know Oracle8i*. This text describes features common to both products, and features that are only available with the Oracle8*i* Enterprise Edition or a particular option.

Overview of the Oracle8*i* Replication API Reference Book

This reference manual describes the Replication Management API. This book assumes that you are familiar with the replication concepts described in the *Oracle8i Replication* manual.

The emphasis of this book is to illustrate how the Replication Management API is used and to serve as a quick reference source for the Replication Management API.

Information in this manual applies to the Oracle8*i* server running on all operating systems.Topics include the following:

- Setup Replication Site
- Create Master Group.
- Create Deployment Template
- Create Snapshot Group
- Conflict Resolution
- Managing Replication Environment
- Replication Management API Reference
- Data Dictionary Views

Audience

This manual is written for application developers and database administrators who develop and maintain Oracle8*i* replication environments.

Knowledge Assumed of the Reader

This manual assumes you are familiar with relational database concepts, distributed database administration, PL/SQL (if using procedural replication), and the operating system under which you run an Oracle replicated environment.

This manual also assumes that you have read and understand the information in the following documents:

- *Oracle8i Concepts*.
- *Oracle8i Administrator's Guide*.
- *Oracle8i Distributed Database Systems*.
- *PL/SQL User's Guide and Reference* (if you plan to use procedural replication).
- *Oracle8i Replication*

How The Oracle8*i* Replication API Reference Book Is Organized

Chapter 1, "Replication Overview"
Provide a "high level" overview of the process required to build a replicated environment. This chapter also contains some prerequisite information required for building a replicated environment.

Chapter 2, "Create Replication Site"
Chapter 2 describes in detail the process of setting up both a master and snapshot site. This chapter should be consulted when building a new replicated environment and when adding either a new master or snapshot site to an established replicated environment.

Chapter 3, "Create a Master Group"
Describes how to build a master group for multimaster replication or as a master for a snapshot site. Chapter 3 builds a master group that replicates data between the three master sites that were set up during Chapter 2.

Chapter 4, "Create Deployment Template"
Deployment templates are the most effective method of distributing a snapshot environment to any number of snapshot site.

Chapter 5, "Create a Snapshot Group"
If deployment templates do not meet your requirements, Chapter 5 describes in detail how to build a snapshot environment at the snapshot site.

Chapter 6, "Conflict Resolution"
The conflict resolution methods described in Chapter 6 will help your data converge at all site when a data conflict arises.

Chapter 7, "Manage Replicated Environment with APIs"
Chapter 7 describes many of the management tasks that you may need to perform to manage your replicated environment. Topics discussed included master group management, altering replicated objects, offline instantiation, and more.

Chapter 8, "Replication Management API Reference"
Describes parameters for packaged procedures used to implement a replicated environment as well as exceptions these procedures might raise.

Chapter 9, "Data Dictionary Views"
Describes views of interest to users of deferred transactions, read-only snapshots, and the advanced replication facility.

Conventions Used in This Manual

This manual uses different fonts to represent different types of information.

Special Notes

Special notes alert you to particular information within the body of this manual:

Note: Indicates special or auxiliary information.

Attention: Indicates items of particular importance about matters requiring special attention or caution.

Additional Information: Indicates where to get more information.

Text of the Manual

The following sections describe conventions used this manual.

UPPERCASE Characters

Uppercase text is used to call attention to command keywords, object names, parameters, filenames, and so on.

For example, "If you create a private rollback segment, the name must be included in the ROLLBACK_SEGMENTS parameter of the parameter file".

Italicized Characters

Italicized words within text indicate the definition of a word, book titles, or emphasized words.

An example of a definition is the following: "A *database* is a collection of data to be treated as a unit. The general purpose of a database is to store and retrieve related information".

An example of a reference to another book is the following: "For more information, see the book *Oracle8i Tuning*."

An example of an emphasized word is the following: "You *must* back up your database regularly".

Code Examples

SQL, Server Manager line mode, and SQL*Plus commands/statements appear separated from the text of paragraphs in a monospaced font. For example:

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');
```

```
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

Example statements may include punctuation, such as commas or quotation marks. All punctuation in example statements is required. All example statements terminate with a semicolon (;). Depending on the application, a semicolon or other terminator may or may not be required to end a statement.

Uppercase words in example statements indicate the keywords within Oracle SQL. When issuing statements, however, keywords are not case sensitive.

Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.

Your Comments Are Welcome

We value your comments as an Oracle user and reader of our manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. This manual contains a Reader's Comment Form that we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. Please mail comments to:

Oracle8*i* Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

You can send comments and suggestions about this manual to the Information Development department at the following e-mail address:

infodev@us.oracle.com

1

Replication Overview

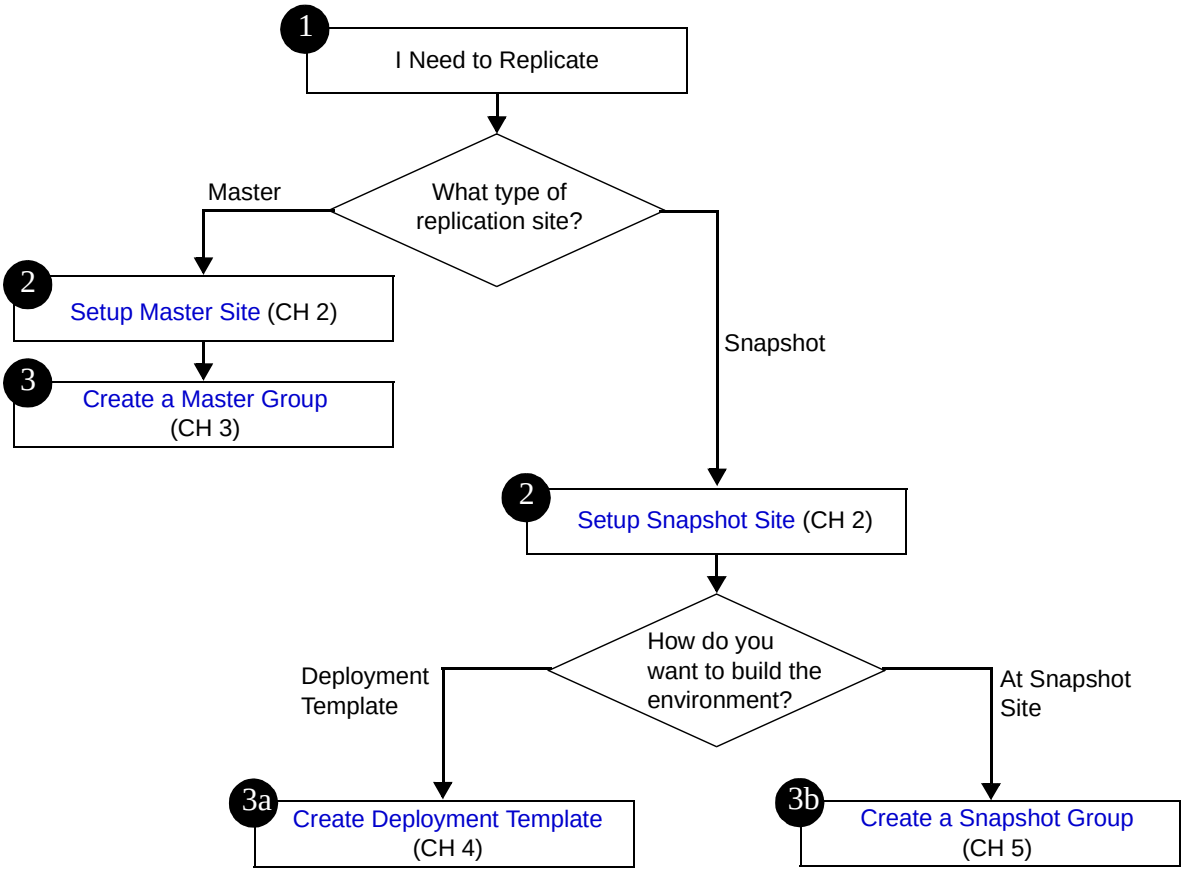
This chapter reviews the process of building a replicated environment. The following will be discussed:

- Creating a Replicated Environment Overview
- Before You Start

Creating a Replicated Environment Overview

Figure 1–1 illustrates the basic steps in building a replicated environment. Regardless of the type of replication site or sites that you are building, you begin by setting up the replicated site.

Figure 1–1 Create Replicated Environment Process



After you have setup your replication sites, you are ready to begin building your master and snapshot groups. After you have built your replication environment, make sure that you review chapters 6 and 7 to learn about conflict resolution and managing your replicated environment.

Before You Start

Before you begin setting up your replication site, there are several items that you need to verify:

- Ensure that global_names is set to TRUE in your INIT.ORA file.
- Ensure that you have allocated enough job processes at each master site.

Global Names

To ensure that your replicated environment will work properly, you must set the global_names parameter in the INIT.ORA file to TRUE. You will see the following description and parameter setting in your INIT.ORA file:

```
# Global Naming -- enforce that a dblink has same name as the db it connects to
global_names = TRUE
```

This parameter must be set to TRUE for each database that is participating in your replicated environment (both master and snapshot sites).

Job Processes

It is important that you have allocated sufficient job processes (sometime referred to as SNP background processes) to handle the automation of your replicated environment (i.e. automatically propagating the deferred transaction queue, purging the deferred transaction queue, refreshing snapshots, etc.).

For multimaster replication, each site will have a scheduled link to each of the other master sites (for example, if you have 6 master sites, each site will have scheduled links to the other 5 sites). You will typically need 1 process for each scheduled link. You may also want to add an additional job process for purging the deferred transaction queue and other user-defined jobs.

By the nature of snapshot replication, each snapshot site will typically have 1 scheduled link to the master database and will require at least 1 job process. Snapshot sites will typically require 1-3 job processes (depending on purge scheduling, user-defined jobs, and the scheduled link). Alternatively, if your users will be responsible for manually refreshing the snapshot through an application interface, you will not need to create a scheduled link and your snapshot site will require 1 less job process.

In addition to defining the amount of job processes, you will also need to define the job interval. The job interval determines how often your job processes "wake-up" to execute any pending operations (such as pushing a queue). While the default value

of 60 seconds is adequate for most replicated environments, you may need to adjust this value to maximize performance for you individual requirements. For example, if you want to propagate changes every 20 seconds, a job interval of 60 seconds would not be sufficient. On the other hand, if you need to propagate your changes once a day, you may only want your SNP process to check for a pending operation once an hour.

The job processes are also defined in the `INIT.ORA` file, usually under the `Oracle replication` heading.

```
job_queue_processes = 7
job_queue_interval = 60
```

After you have modified the contents of your INIT.ORA file, you will need to restart your database with these new settings (see *Oracle8i Administrator's Guide* for information on restarting your database).

2

Create Replication Site

This chapter illustrates how to setup both a master and a snapshot replication site using the Replication Management API. The following topics will be discussed:

- Setting Up a Replication Site Overview
- Setup Master Site
- Setup Snapshot Site

Setting Up a Replication Site Overview

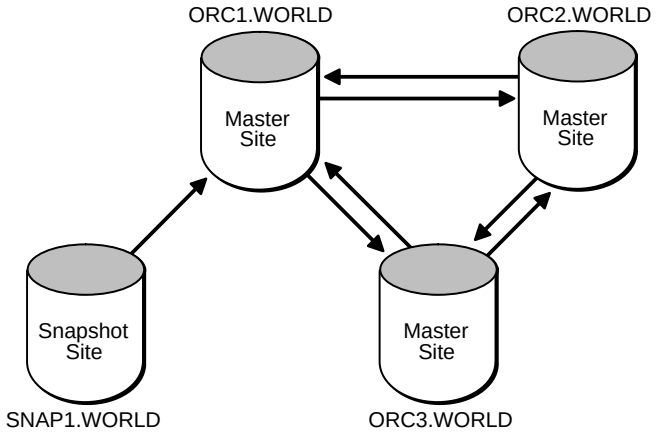
Before you begin building your replicated environment, you need to setup the sites that will participate in the replicated environment. As illustrated in Figure 2–2 and Figure 2–3, there are separate processes for setting up a master site versus setting up a snapshot site.

This chapter assumes that you have the following:

- 4 Databases:
  - ORC1.WORLD
  - ORC2.WORLD
  - ORC3.WORLD
  - SNAP1.WORLD

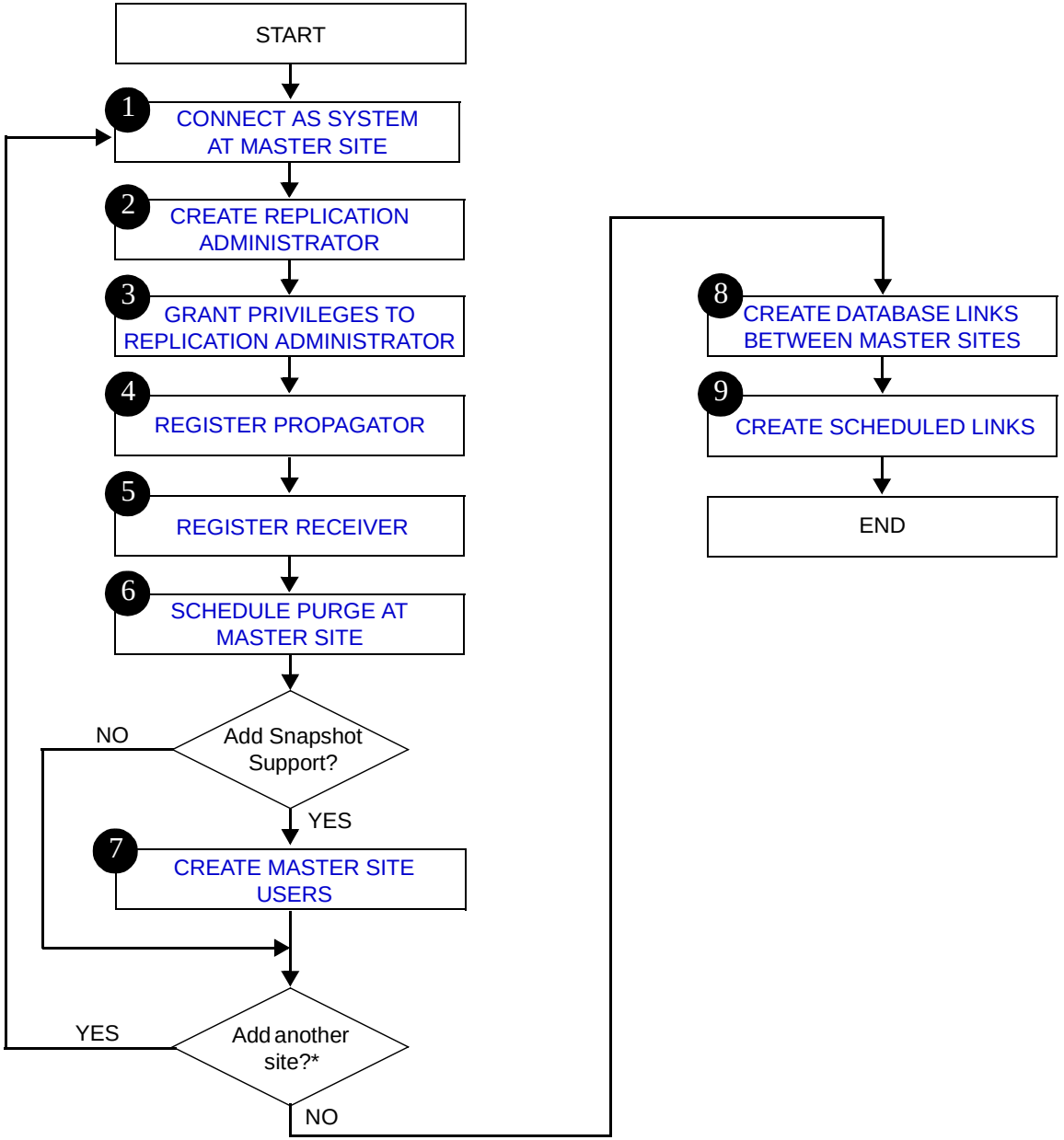
Chapters 2 — 6 will work with the replication environment illustrated in Figure 2–1 that you will begin to create during this chapter.

Figure 2–1 Three Master Sites and One Snapshot Site



You will need to follow the procedures identified in Figure 2–2 when you build a new master site or in Figure 2–3 when you build a new snapshot site.

Figure 2–2 Setup Master Sites



*Multiple master sites (multimaster replication) can only be used with Oracle8*i* Enterprise Edition.

Setup Master Site

```
/*************************************************************************
STEP 1 @ ORC1.WORLD:
CONNECT AS SYSTEM AT MASTER SITE
*************************************************************************/

--You will need to connect as SYSTEM to the database that you want to
--setup for replication. After you setup ORC1.WORLD, you will need to
--begin again with STEP 1 for sites ORC2.WORLD on page 2-7 and
--ORC3.WORLD on page 2-10.

CONNECT system/manager@orc1.world

/*************************************************************************
STEP 2 @ ORC1.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator will be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator needs to be created at each database that will participate
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 3 @ ORC1.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-168.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API will grant the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/

--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.
```

```
/*************************************************************************
STEP 4 @ ORC1.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see
"REGISTER_PROPAGATOR procedure" on page 8-34.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/

/*************************************************************************
STEP 5 @ ORC1.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API,
see "REGISTER_USER_REPGROUP procedure" on page 8-170.
*************************************************************************/

--The receiver will receive the propagated deferred transactions sent
--by the propagator from other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/
```

```
/*************************************************************************
STEP 6 @ ORC1.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-35.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API will automate the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/

/*************************************************************************
STEP 7:
CREATE MASTER SITE USERS
*************************************************************************/

--STEP 7a: CREATE PROXY SNAPSHOT ADMINISTRATOR
--The proxy snapshot administrator performs tasks at the target master
--site on behalf of the snapshot administrator at the snapshot
--site.  See "Security Setup for Snapshot Replication" in the
--Oracle8i Replication manual.

CONNECT system/manager@orc1.world

CREATE USER proxy_snapadmin IDENTIFIED BY proxy_snapadmin;

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'proxy_snapadmin',
      PRIVILEGE_TYPE => 'proxy_snapadmin',
      LIST_OF_GNAMES => NULL);
END;
/
```

```
--STEP 7b: CREATE PROXY REFRESHER
--The proxy refresher performs tasks at the master site on behalf of
--the refresher at the snapshot site.

CREATE USER proxy_refresher IDENTIFIED BY proxy_refresher;

GRANT CREATE SESSION TO proxy_refresher;
GRANT SELECT ANY TABLE TO proxy_refresher;

/*************************************************************************
STEP 1  @ ORC2.WORLD:
CONNECT AS SYSTEM
*************************************************************************/

--NOTE:
--Multiple master sites (multimaster replication) can only be used with
--Oracle8i Enterprise Edition. If you are not using Oracle8i Enterprise
--Edition, skip to step 8 on page 2-12.

--You will need to connect as SYSTEM to the database that you want to
--setup for replication. After you setup ORC2.WORLD, you will need to
--begin again with STEP 1 for site ORC3.WORLD on page 2-10.

CONNECT system/manager@orc2.world

/*************************************************************************
STEP 2  @ ORC2.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator will be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator needs to be created at each database that will participate
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;
```

```
/*************************************************************************
STEP 3  @ ORC2.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-168.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API will grant the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/

--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.
/*************************************************************************
STEP 4  @ ORC2.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see "REGISTER_
PROPAGATOR procedure" on page 8-34.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/
```

```
/*************************************************************************
STEP 5  @ ORC2.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API, see "REGISTER_
USER_REPGROUP procedure" on page 8-170.
*************************************************************************/

--The receiver will receive the propagated deferred transactions sent
--by the propagator from the other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/

/*************************************************************************
STEP 6  @ ORC2.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-35.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API will automate the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/
```

```
/*************************************************************************
STEP 1  @ ORC3.WORLD:
CONNECT AS SYSTEM
*************************************************************************/

--NOTE:
--Multiple master sites (multimaster replication) can only be used with
--Oracle8i Enterprise Edition. If you are not using Oracle8i Enterprise
--Edition, skip to step 8 on page 2-12.

--You will need to connect as SYSTEM to the database that you want to
--setup for replication.

CONNECT system/manager@orc3.world

/*************************************************************************
STEP 2 @ ORC3.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator will be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator needs to be created at each database that will participate
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 3 @ ORC3.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-168.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API will grant the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/
```

```
--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.

/*************************************************************************
STEP 4 @ ORC3.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see "REGISTER_
PROPAGATOR procedure" on page 8-34.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/

/*************************************************************************
STEP 5 @ ORC3.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API, see "REGISTER_
USER_REPGROUP procedure" on page 8-170.
*************************************************************************/

--The receiver will receive the propagated deferred transactions sent
--by the propagator from the other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/
```

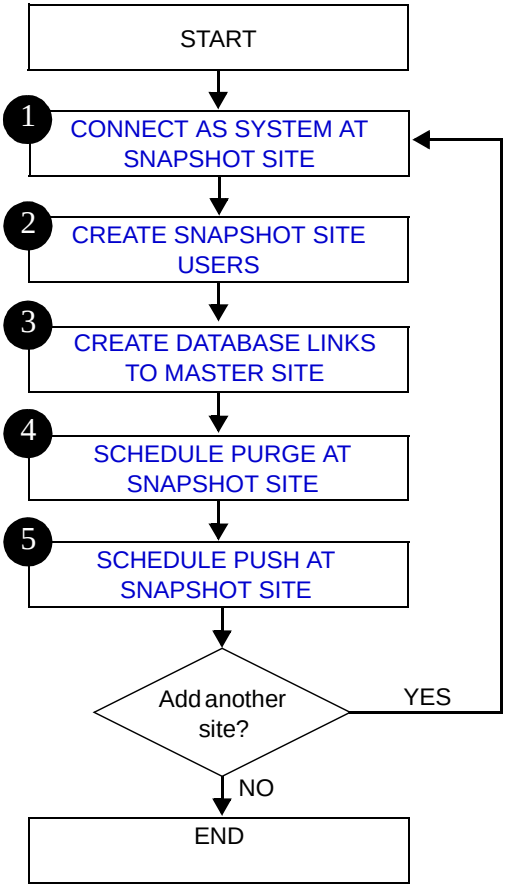
```
/*************************************************************************
STEP 6 @ ORC3.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-35.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API will automate the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/

/*************************************************************************
STEP 8:
CREATE DATABASE LINKS BETWEEN MASTER SITES

The database links provide the necessary distributed mechanisms to allow
the different replication sites to replicate data between themselves. See
the Oracle8i Distributed Database Systems for more information.
*************************************************************************/

--Before you create any private database links, you need to create the
--public database links that each private database link will use.
--You will then need to create a database link between all replication
--administrators at each of the master sites that you have setup.

CONNECT system/manager@orc1.world
CREATE PUBLIC DATABASE LINK orc2.world USING 'orc2.world';
CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';

CONNECT repadmin/repadmin@orc1.world
CREATE DATABASE LINK orc2.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc3.world CONNECT TO repadmin IDENTIFIED BY repadmin;
```

```
CONNECT system/manager@orc2.world
CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';
CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';

CONNECT repadmin/repadmin@orc2.world
CREATE DATABASE LINK orc1.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc3.world CONNECT TO repadmin IDENTIFIED BY repadmin;

CONNECT system/manager@orc3.world
CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';
CREATE PUBLIC DATABASE LINK orc2.world USING 'orc2.world';

CONNECT repadmin/repadmin@orc3.world
CREATE DATABASE LINK orc1.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc2.world CONNECT TO repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 9:
CREATE SCHEDULED LINKS

Create a scheduled link by defining a database link when you execute the
SCHEDULE_PUSH procedure (see "SCHEDULE_PUSH procedure" on page 8-37
for more information).
*************************************************************************/

--The scheduled link determines how often your deferred transaction queue is
--propagated to each of the other master sites. You need to execute the
--SCHEDULE_PUSH procedure for each database link that you created
--in STEP 7; the database link is specified in the DESTINATION parameter
--of the SCHEDULE_PUSH procedure.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc2.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/
```

```
BEGIN
DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc3.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc3.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc2.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/
```

Setup Snapshot Site

Figure 2–3 Setup Snapshot Sites



```
/*************************************************************************
STEP 1:
CONNECT AS SYSTEM AT SNAPSHOT SITE
*************************************************************************/
--You will need to connect as SYSTEM to the database that you want to
--setup as a snapshot site.

CONNECT system/manager@snap1.world

/*************************************************************************
STEP 2:
CREATE SNAPSHOT SITE USERS
*************************************************************************/

--There are several users that need to be created at the snapshot site.
--These users are:
--  SNAPSHOT ADMINISTRATOR
--  PROPAGATOR
--  REFRESHER

--STEP 2a: CREATE SNAPSHOT ADMINISTRATOR
--The snapshot administrator is responsible for creating and managing
--the snapshot site. Execute the GRANT_ADMIN_ANY_SCHEMA
--procedure to grant the snapshot administrator the appropriate privileges.

CREATE USER snapadmin IDENTIFIED BY snapadmin;

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'snapadmin');
END;
/

--STEP 2b: CREATE PROPAGATOR
--The propagator is responsible for propagating the deferred transaction
--queue to the target master site.

CREATE USER propagator IDENTIFIED BY propagator;

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'propagator');
END;
/
```

```
--STEP 2c: CREATE REFRESHER
--The refresher is responsible for "pulling" changes made to the replicated
--tables at the target master site to the snapshot site.

CREATE USER refresher IDENTIFIED BY refresher;

GRANT CREATE SESSION TO refresher;
GRANT ALTER ANY SNAPSHOT TO refresher;

/*************************************************************************
STEP 3:
CREATE DATABASE LINKS TO MASTER SITE
*************************************************************************/

--STEP 3A: CREATE PUBLIC DATABASE LINK

CONNECT system/manager@snap1.world

CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';

--STEP 3b: CREATE SNAPSHOT ADMINISTRATOR DATABASE LINK
--You need to create a database link from the snapshot administrator at
--the snapshot site to the proxy snapshot administrator at
--the master site.

CONNECT snapadmin/snapadmin@snap1.world;

CREATE DATABASE LINK orc1.world
  CONNECT TO proxy_snapadmin IDENTIFIED BY proxy_snapadmin;

--STEP 3c: CREATE PROPAGATOR/RECEIVER DATABASE LINK
--You need to create a database link from the propagator at the
--snapshot site to the receiver at the master site (the receiver was defined
--when you created the master group - see "REGISTER RECEIVER" on page 2-5
--for more information).

CONNECT propagator/propagator@snap1.world

CREATE DATABASE LINK orc1.world
  CONNECT TO repadmin IDENTIFIED BY repadmin;
```

```
/*************************************************************************
STEP 4:
SCHEDULE PURGE AT SNAPSHOT SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-35.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API will automate the purge process for you. If your snapshot
--site will only contain "read-only" snapshots, then you will not need to
--execute this procedure.

CONNECT snapadmin/snapadmin@snap1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
   NEXT_DATE => SYSDATE,
   INTERVAL => 'SYSDATE + 1/24',
   DELAY_SECONDS => 0,
   ROLLBACK_SEGMENT => '');
END;
/

/*************************************************************************
STEP 5:
SCHEDULE PUSH AT SNAPSHOT SITE

For additional information about the SCHEDULE_PUSH API, see "SCHEDULE_PUSH
procedure" on page 8-37.
*************************************************************************/

--The SCHEDULE_PUSH API schedules when the deferred transaction queue
--should be propagated to the target master site.

CONNECT snapadmin/snapadmin@snap1.world
```

```
BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 1/24',
      NEXT_DATE => SYSDATE,
      STOP_ON_ERROR => FALSE,
      DELAY_SECONDS => 0,
      PARALLELISM => 0);
END;
/
```

3

Create a Master Group

This chapter illustrates how to create a master group at a master replication site. The following topics will be discussed:

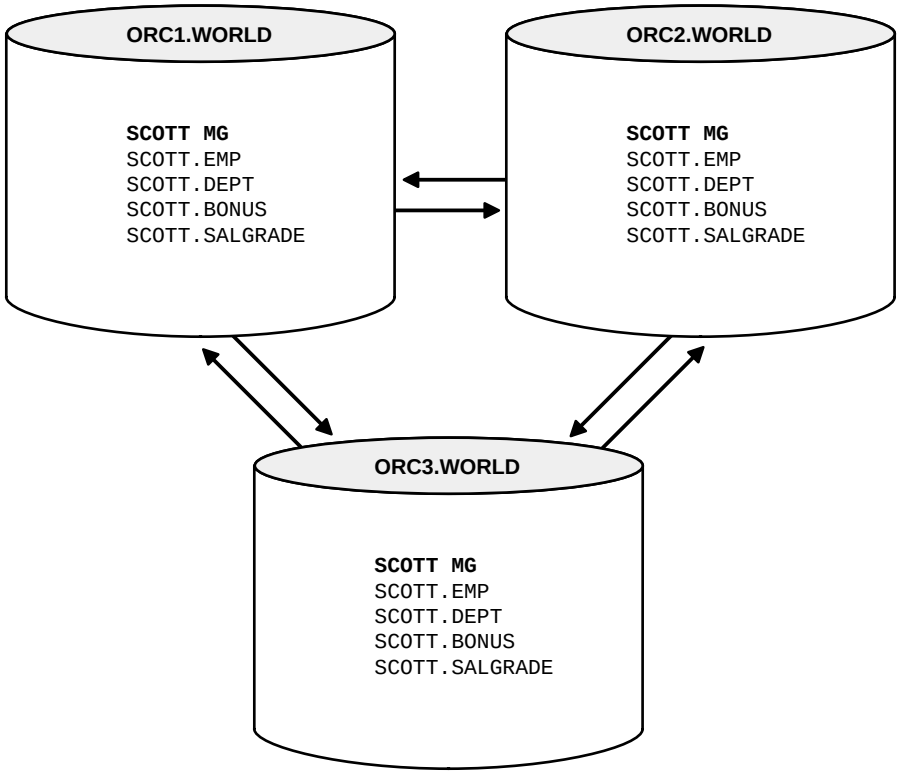
- Creating a Master Group Overview
- Create Master Group

Creating a Master Group Overview

After you have setup your master site (see "Create Replication Site" on page 2-1 for details), you are ready to begin building a master group. As illustrated in Figure 3–2, there is a distinct sequence that you need to follow to successfully build a replicated environment.

This chapter will create the SCOTT_MG master group and will replicate the objects illustrated in Figure 3–1:

Figure 3–1 Replicate EMP, DEPT, BONUS, and SALGRADE between all sites.



Before You Start

In order for the script in this chapter to work as designed, it is assumed that the schema SCOTT exists at ORC1.WORLD (and optionally ORC2.WORLD and ORC3.WORLD) and contains the following objects:

- EMP
- DEPT
- BONUS
- SALGRADE

If you don't have the SCOTT schema at ORC1.WORLD or the SCOTT objects do not exist, you can run a script that comes with your Oracle database to create the sample schema SCOTT and the corresponding objects.

Complete the following:

1. Connect to ORC1.WORLD as user SYSTEM.

   ```
   CONNECT system/manager@orc1.world
   ```

2. If the SCOTT schema does exist, skip to step 3. Otherwise, create the user SCOTT as illustrated below:

   ```
   CREATE USER scott IDENTIFIED BY tiger;
   ```

3. Run the SCOTT.SQL script that is contained in your <ORACLE_HOME>\RDBMS\ADMIN directory.

   The schema SCOTT must exist (and be IDENTIFIED BY tiger) in order for this script to run properly. If it does not exist, be sure that you complete step 2.

   Note: If you are running multiple database instances on the same computer, you may need to alter the CONNECT string contained within the SCOTT.SQL script to contain the target database. For example, you would replace

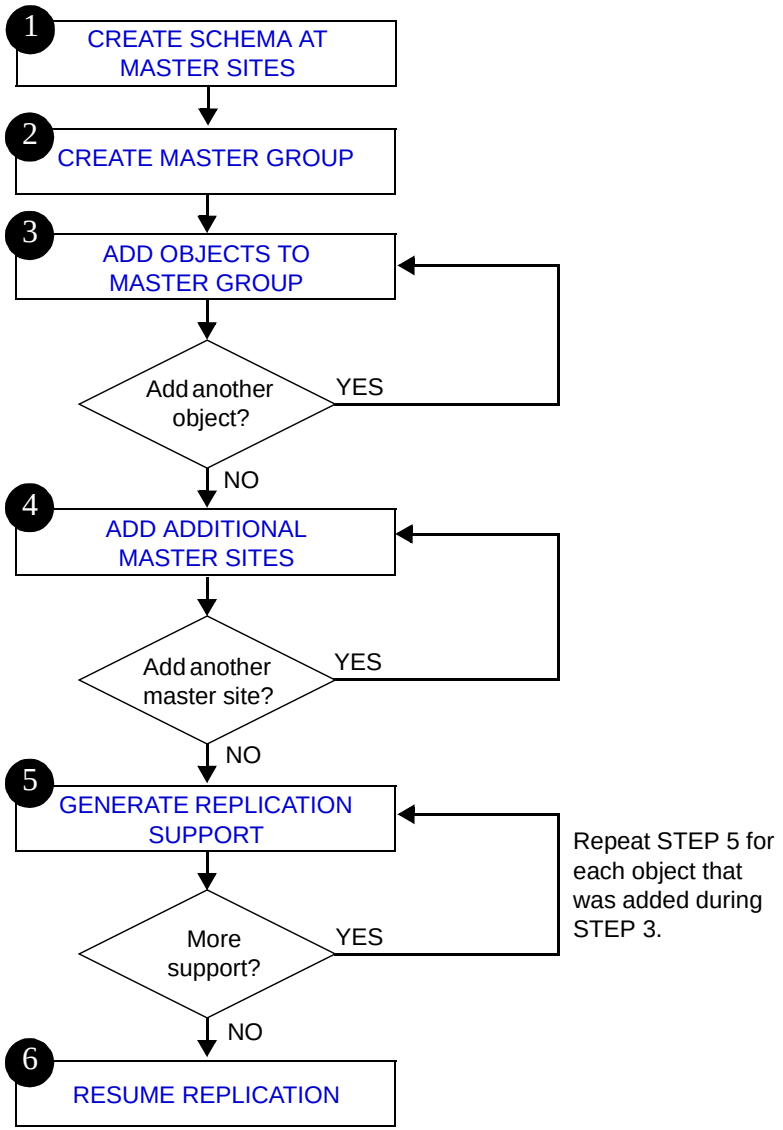
   ```
   CONNECT scott/tiger
   ```

   with

   ```
   CONNECT scott/tiger@orc1.world
   ```

After you have completed the three steps above, you will have a "fresh" copy of the EMP, DEPT, BONUS, and SALGRADE tables. The tables EMP and DEPT will have a primary key created during this script, though BONUS and SALGRADE will not have a primary key.

Figure 3–2 Create Master Group



Create Master Group

```
/*************************************************************************
STEP 1:
CREATE SCHEMA AT MASTER SITES
*************************************************************************/

CONNECT system/manager@orc2.world;
CREATE USER scott IDENTIFIED BY tiger;
GRANT CONNECT, RESOURCE TO scott;

CONNECT system/manager@orc3.world;
CREATE USER scott IDENTIFIED BY tiger;
GRANT CONNECT, RESOURCE TO scott;

/*************************************************************************
STEP 2:
CREATE MASTER GROUP
*************************************************************************/

--Use the CREATE_MASTER_REPGROUP API to define a new master group.
--When you add an object to your master group or perform other replication
--administrative tasks, you will reference the master group name defined
--during this step. The following must be executed by the replication
--administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPGROUP (
      GNAME => 'scott_mg');
END;
/

/*************************************************************************
STEP 3:
ADD OBJECTS TO MASTER GROUP
*************************************************************************/

--Use the CREATE_MASTER_REPOBJECT API to an object to your master group.
--In most cases, you will probably be adding tables to your master group,
--but you can also add indexes, procedures, views, synonyms, etc.  See
--CREATE_MASTER_REPOBJECT procedure on page 8-110 for additional
--information.
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'emp',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'dept',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

--If you will recall from the "Before You Start" section on page 3-3, there
--is no primary key for the BONUS or SALGRADE tables. In order for
--replication to work properly, the replicated table either needs a
--primary key or to have a "set column." The DBMS_REPCAT.SET_COLUMNS procedure
is
--sufficient for multimaster replication only, but if you will also
--support fast refreshable snapshots, you will need a primary key.
--It is easier to alter your object before you add it to your master group.

ALTER TABLE scott.bonus ADD (CONSTRAINT bonus_pk PRIMARY KEY(ename));

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'bonus',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/
```

```
--You will need to modify the SCOTT.SALGRADE object just as you altered the
--SCOTT.BONUS object in the previous step..

ALTER TABLE scott.salgrade ADD (CONSTRAINT salgrade_pk PRIMARY KEY(grade));

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'salgrade',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

/*************************************************************************
STEP 4:
ADD ADDITIONAL MASTER SITES
*************************************************************************/

--After you have defined your master group at the MASTERDEF site (the
--site where the master group was created becomes the MASTER DEFINITION
--site by default), you can define the other sites that will participate
--in the replicated environment. You might have guessed that we will be
--adding the ORC2.WORLD and ORC3.WORLD sites to our replicated environment.

BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc2.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*
NOTE: You should wait until ORC2.WORLD appears in the DBA_REPSITES view
before continuing. Execute the following SELECT statement in another
SQL*Plus session to make sure that ORC2.WORLD has appeared):

SELECT * FROM dba_repsites WHERE gname = 'scott_mg';
*/
PAUSE Press <RETURN> to continue.
```

```
BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc3.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*
NOTE: You should wait until ORC3.WORLD appears in the DBA_REPSITES view
before continuing. Execute the following SELECT statement in another
SQL*Plus session to make sure that ORC3.WORLD has appeared):

SELECT * FROM dba_repsites WHERE gname = 'scott_mg';
*/
PAUSE Press <RETURN> to continue.

/*****************************************************************************
STEP 5:
GENERATE REPLICATION SUPPORT
*****************************************************************************/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'dept',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

/*
NOTE: You should wait until the DBA_REPCATLOG view is empty before
resuming master activity. Execute the following SELECT statement
to monitor your DBA_REPCATLOG view:

SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';
*/
PAUSE Press <RETURN> to continue.

/*************************************************************************
STEP 6:
RESUME REPLICATION
*************************************************************************/

--After you have completed creating your master group, adding replication
--objects, generating replication support, and adding additional master
--databases, you need to resume replication activity. The
--RESUME_MASTER_ACTIVITY procedure API will "turn on" replication for
--the specified master group.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

4

Create Deployment Template

This chapter illustrates how to build a deployment template using the Replication Managment API. The following topics will be discussed:

- Oracle Deployment Templates Concepts
- Build Deployment Template
- Package for Instantiation
- Instantiate Deployment Template

Oracle Deployment Templates Concepts

Oracle offers Deployment Templates to allow the database administrator to package a snapshot environment for easy, custom, and secure distribution/installation. A deployment template can be as simple as containing a single snapshot with a fixed data set, or as complex as hundreds of snapshots with a dynamic data set based on one or more variables. The goal is to create the environment once and deploy the deployment template as often as necessary. Oracle deployment templates feature:

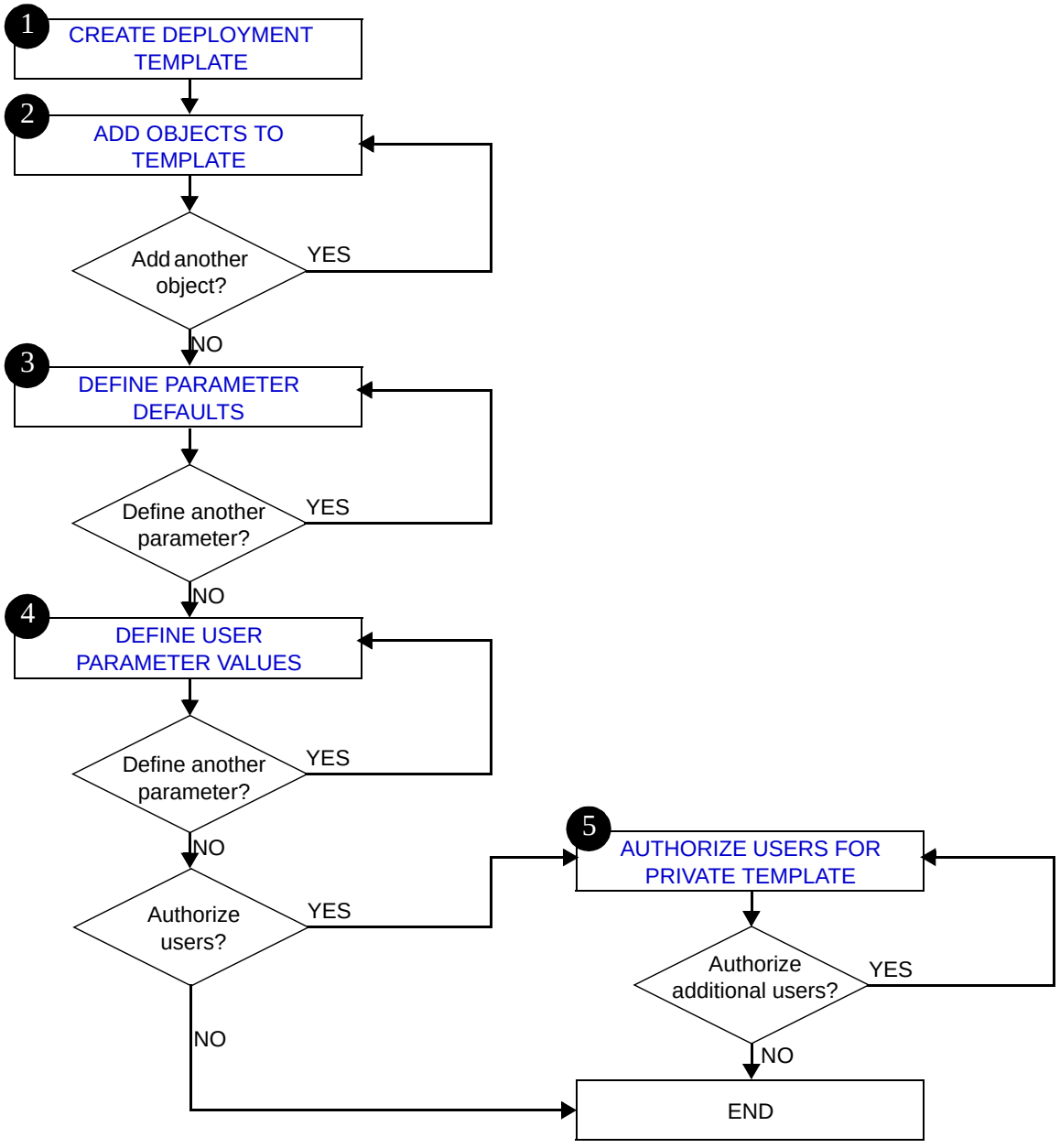
- Centralized control
- Capability to repeatedly deploy snapshot environment
- Template parameters allow data subsetting at remote site
- Authorized user list to control template instantiation and data access

To prepare a snapshot environment for deployment, the DBA creates a *template* at the master site. This template will store all of the information needed to deploy a snapshot environment, including the DDL to create the objects at the remote site and the target refresh group. This template also maintains links to user security information and template parameters for custom snapshot creation.

Build Deployment Template

This section contains a complete script example of how to construct a deployment template using the Replication Management API. For deployment template conceptual and architectural information, be sure to read Chapter 4 in the *Oracle8i Replication* manual.

Figure 4–1 Setup Deployment Template



Be sure to read the comments contained within the scripts, as they contain important and useful information about building templates with the Replication Management API.

> **Note:** Vertical partitioning is not supported using the Replication Management API. See "Vertical Partitioning" in the *Oracle8i Replication* manual for more information.

```
--This script creates a deployment template that contains
--4 template objects, two template parameters, a set of user
--parameter values, and an authorized user.  A template is
--built in the following order:
--
--STEP 1: Define Refresh Group Template
--STEP 2: Add template objects to DT_PERSONNEL
--STEP 3: Define Parameter Defaults and Prompt Text
--STEP 4: Define User Parameter Values
--STEP 5: Authorize Users for Private Template


CONNECT repadmin/repadmin@orc3.world

/*************************************************************************
STEP 1:
CREATE DEPLOYMENT TEMPLATE
*************************************************************************/

--Before you begin assembling the components of your deployment
--template, you need to use the CREATE_RERESH_TEMPLATE procdure
--to define the name of your deployment template, along with
--several other template characteristics (Public/Private status,
--target refresh group, and owner).

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
           OWNER => 'scott',
           REFRESH_GROUP_NAME => 'personnel',
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           TEMPLATE_COMMENT => 'personnel deployment template',
           PUBLIC_TEMPLATE => 'N');
END;
/
```

```
/*************************************************************************
STEP 2:
ADD OBJECTS TO TEMPLATE
*************************************************************************/

--STEP 2a: Create EMP Snapshot

--You will notice that the following procedure uses the DBMS_LOB
--package. This package is required to insert values into the
--DDL_TEXT parameter of the CREATE_TEMPLATE_OBJECT function, which
--has a CLOB datatype.  You will see the DBMS_LOB package used
--whenever a value needs to be inserted into a CLOB parameter.
--For more information about using the DBMS_LOB package and LOBs
--in general, see the Oracle8i Application Developer's Guide - Fundamentals.

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_emp AS SELECT
      empno, ename, job, mgr, hiredate, sal, comm, deptno
      FROM scott.emp@&dblink WHERE deptno = &dept';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_emp',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           master_rollback_seg => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
--STEP 2b: Create DEPT Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_dept AS SELECT
      deptno, dname, loc
      FROM scott.dept@&dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_dept',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

--STEP 2c: Create SALGRADE Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_salgrade AS SELECT
      grade, losal, hisal
      FROM scott.salgrade@&dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_salgrade',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
--STEP 2d: Create BONUS Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_bonus AS SELECT
      ename, job, sal, comm
      FROM scott.bonus@&dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_bonus',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

/*************************************************************************
STEP 3:
DEFINE PARAMETER DEFAULTS
*************************************************************************/

--Unlike using the "CREATE" functions and procudres in the other
--steps, you will use the ALTER_TEMPLATE_PARM procedure to define
--a template parameter value and prompt string. You will use the
--"ALTER" procedure because the actual parameter was created in
--step 2; recall that you defined the &DBLINK and &DEPT parameters
--in the DDL_TEXT parameter. Oracle detects these parameters in
--the DDL and automatically creates the template parameter. Use
--the ALTER_TEMPLATE_PARM procdure to define the remainder of the
--template parameter information (i.e. default parameter value and
--prompt string).
```

```
--STEP 3a: DEPT Parameter

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := '20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      REFRESH_TEMPLATE_NAME => 'dt_personnel',
      PARAMETER_NAME => 'dept',
      NEW_DEFAULT_PARM_VALUE => templob,
      NEW_PROMPT_STRING => 'Enter your department number:',
      NEW_USER_OVERRIDE => 'Y');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

--STEP 3b: DBLINK Parameter

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'ORC2.WORLD';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      REFRESH_TEMPLATE_NAME => 'dt_personnel',
      PARAMETER_NAME => 'dblink',
      NEW_DEFAULT_PARM_VALUE => templob,
      NEW_PROMPT_STRING => 'Enter target database link:',
      NEW_USER_OVERRIDE => 'N');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
/*************************************************************************
STEP 4:
DEFINE USER PARAMETER VALUES
*************************************************************************/

--To automate the instantiation of custom data sets at
--individual remote snapshot sites, you can define USER
--PARAMETER values that will automatically be used when
--the specified user instantiates the target template.
--The CREATE_USER_PARM_VALUE enables you to assign a
--parameter value to a specific user.

--STEP 4a: Define User Parameter Value for user SCOTT

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := '30';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           PARAMETER_NAME => 'dept',
           USER_NAME => 'scott',
           PARM_VALUE => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
--STEP 4b: Define User Parameter Value for user SCOTT

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'ORC2.WORLD';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           PARAMETER_NAME => 'dblink',
           USER_NAME => 'scott',
           PARM_VALUE => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

/*****************************************************************************
STEP 5:
AUTHORIZE USERS FOR PRIVATE TEMPLATE
*****************************************************************************/

--Since this is a private template (PUBLIC_TEMPLATE => 'N'
--in the DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE function
--defined in STEP 1), you need to authorize users to
--instantiate the DT_PERSONNEL deployment template. Use
--the DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION function
--to create authorized users.

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION (
           USER_NAME => 'scott',
           REFRESH_TEMPLATE_NAME => 'dt_personnel');
END;
/

COMMIT;
```

Package for Instantiation

After you have completed building your deployment template, you need to package the template for instantiation. This example will illustrate how to use both the online and offline instantiation procedures (you will notice that the instantiation procedures are very similar, you simply use either the INSTANTIATE_ONLINE function or INSTANTIATE_OFFLINE function according to your needs). This section will accomplish two tasks: create the instantiation script and save the instantiation script to a file.

Create Instantiation Script

When you execute either the INSTANTIATE_ONLINE or INSTANTIATE_OFFLINE functions, Oracle populates the USER_REPCAT_TEMP_OUTPUT view with the script to create the remote snapshot environment. The difference to remember is that an offline instantiation script contains the DDL and the DML to create both the snapshot environment and populate it with the appropriate data set. An online instantiation contains only the DDL to create the snapshot environment; the environment is populated with the data when the script is executed at the remote snapshot site (this requires a connection to the master site).

Complete the steps in either the "Offline Instantiation Package" or "Online Instantiation Package" according to your needs.

Offline Instantiation Package

The INSTANTIATE_OFFLINE function creates a script that creates the snapshot environment according to the contents of a specified deployment template. In addition to containing the DDL to create the snapshot environment, this script also contains the DML to populate the snapshot environment with the appropriate data set.

Note: If you are packaging your template at the same master site that contains the target master objects for your deployment template, you will need to create a loopback database link.

```
--Use the INSTANTIATE_OFFLINE function to "package" the
--template for offline instantiation by a remote snapshot
--site. Executing this procedure will create a script that
--creates that snapshot environment, as well as populate the
--environment with the proper data set; this script is stored
--in the temporary USER_REPCAT_TEMP_OUTPUT view.
```

```
SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
               REFRESH_TEMPLATE_NAME => 'dt_personnel',
               USER_NAME => 'scott',
               SITE_NAME => 'la_regional',
               NEXT_DATE => SYSDATE,
               INTERVAL => 'SYSDATE + (1/144)');
   DBMS_OUTPUT.PUT_LINE('Template ID = ' || dt_num);
END;
/
COMMIT;
/
```

Be sure to note the number that is returned for the DT_NUM variable. You will be required to use this number when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated script. Be sure that you complete the steps in "Save Instantiation Script to File" after you complete this section.

Online Instantiation Package

The INSTANTIATE_ONLINE function creates a script that creates the snapshot environment according to the contents of a specified deployment template. When this script is executed at the remote snapshot site, Oracle will create the snapshot site according to the DDL in the script and will populate the environment with the appropriate data set from the master site. This requires that the remote snapshot site has a "live" connection to the master site.

For additional requirements, be sure to read "Deploying Template" in the *Oracle8i Replication* manual.

```
--Use the INSTANTIATE_ONLINE function to "package" the
--template for online instantiation by a remote snapshot
--site. Executing this procedure will create a script that
--creates that snapshot environment; this script is stored in
--the temporary USER_REPCAT_TEMP_OUTPUT view.
```

```
SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
               REFRESH_TEMPLATE_NAME => 'dt_personnel',
               USER_NAME => 'scott',
               SITE_NAME => 'snap1.world',
               NEXT_DATE => SYSDATE,
               INTERVAL => 'SYSDATE + (1/144)');
   DBMS_OUTPUT.PUT_LINE('Template ID = ' || dt_num);
END;
/
COMMIT;
/
```

Be sure to note the number that is returned for the DT_NUM variable. You will be required to use this number when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated script. Be sure that you complete the steps in "Save Instantiation Script to File" after you complete this section.

Save Instantiation Script to File

The easiest way to save the contents of the USER_REPCAT_TEMP_OUTPUT view is to use the Server Manager spool feature to save the results of a SELECT statement. Complete the following steps to save your deployment template script to a file:

> **Note:** The following steps must be performed immediately after you have called either the INSTANTIATE_OFFLINE or INSTANTIATE_ONLINE functions (the contents of the USER_REPCAT_TEMP_OUTPUT view are temporary). If you have not completed the steps in the "Create Instantiation Script" section (previous section), do so now and then complete the following steps.

1. Type SPOOL *filename*.sql (where *filename* is the name of your script) at the SVRMGR> prompt and press <ENTER>. For example, you might enter (since you may have to generate many instantiation files, make sure you name your files with an easily recognizable name):

   ```
   SPOOL d:\snap1_world.sql
   ```

Your instantiation script will be saved as `snap1_world.sql` in the `ORACLE_HOME` directory, unless otherwise specified (as in the example above). If necessary, precede the filename with a fully qualified path to save the script to a different directory.

2. Type the following select statement at the `SVRMGR>` prompt and press <ENTER>:

```
SELECT DBMS_REPCAT_RGT.VC2_FROM_CLOB(text) text
FROM user_repcat_temp_output
WHERE output_id = dt_num ORDER BY LINE;
```

dt_num is the value that was returned when you executed the INSTANTIATE_ONLINE or INSTANTIATE_OFFLINE functions (illustrated in "Create Instantiation Script").

3. When Server Manager has completed displaying the contents of the REPCAT$_TEMP_OUTPUT table, type the following at the `SVRMGR>` prompt and press <ENTER>:

```
SPOOL OFF
```

The file that you specified in step 1 will be saved in the directory specified. This file contains the script required to build the snapshot environment.

After you have created the instantiation script and saved it to a file, you will need to distribute this file to the remote snapshot sites that need to instantiate the template. You can distribute this file by posting the file on an FTP site or saving the file to a CD-ROM, floppy disk, or other distribution media.

Instantiate Deployment Template

After the instantiation script has been distributed to the remote snapshot sites, either by FTP, CD-ROM, floppy disk, etc., you are ready to instantiate the deployment template at the remote snapshot site. Be sure to read "Deploying Template" in the *Oracle8i Replication* manual for a list of requirements that your snapshot site must meet before instantiating your deployment template.

This example uses SQL*Plus to instantiate the deployment template. If the remote snapshot site does not have SQL*Plus, see "Deploying Template" in the *Oracle8i Replication* manual for additional instantiation options.

The following script demonstrates how to complete the offline instantiation process at the remote snapshot site:

```
/*************************************************************************
STEP 1:
CREATE SCHEMA AND DATABASE LINKS
*************************************************************************/

--Before you execute the instantiation script at the remote snapshot site,
--you must create the schema that will contain the replicated objects.

CONNECT system/manager@snap1.world

CREATE USER scott IDENTIFIED BY tiger;

GRANT CONNECT, RESOURCE TO scott;

--Before you can create the private database link, you must create a public
--database link.

CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';

--Connect as the target user (wcreekba) and create a private database link
--to the target master site (the target user must also exist at the master
--site).

CONNECT scott/tiger@snap1.world

CREATE DATABASE LINK orc3.world
   CONNECT TO scott IDENTIFIED BY tiger;

/*************************************************************************
STEP 2:
EXECUTE THE OFFLINE INSTANTIATION SCRIPT
*************************************************************************/

RUN D:\snap1_world.sql
```

Depending on the size of the snapshot environment created and the amount of data loaded, the instantiation procedure may take a substantial amount of time.

After Instantiation

Since you have just instantiated a deployment template using the offline instantiation method, you should perform a refresh as soon as possible; complete the following:

```
EXECUTE DBMS_REFRESH.REFRESH('personnel');
```

5

Create a Snapshot Group

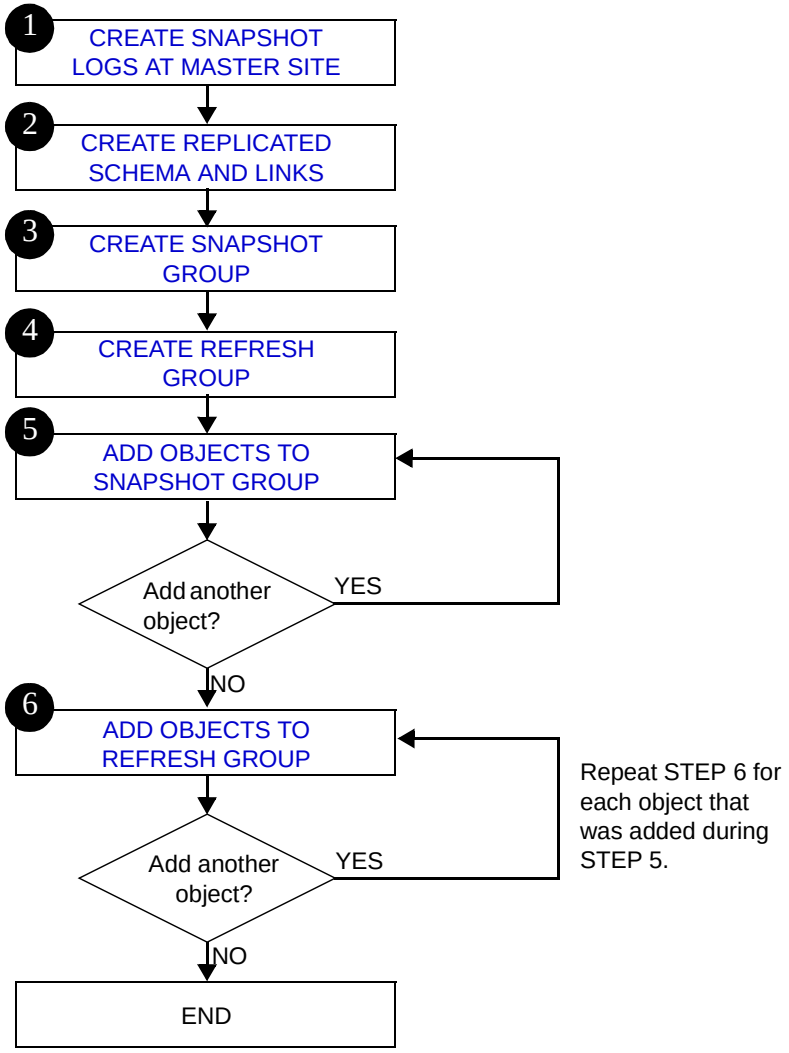
This chapter illustrates how to create a snapshot group at a remote snapshot replication site. The following topics will be discussed:

- Creating a Snapshot Group Overview
- Create Snapshot Group

Creating a Snapshot Group Overview

After you have setup your snapshot site (see Chapter 2, "Create Replication Site") and have created at least one master group (see Chapter 3, "Create a Master Group"), you are ready to begin creating a snapshot group at the remote snapshot site. Figure 5–1 illustrates the process of creating a snapshot group.

Figure 5–1 Setup Snapshot Group



Create Snapshot Group

```
/*************************************************************************
STEP 1:
CREATE SNAPSHOT LOGS AT MASTER SITE

See the CREATE SNAPSHOT LOG in the Oracle8i SQL Reference for
detailed information about this SQL statement.
*************************************************************************/

--If you want one of your master sites to support a snapshot site, then
--you need to create snapshot logs for each master table that will be
--replicated to a snapshot. If you'll recall from Figure 2–1 on page 2-2,
--ORC1.WORLD will serve as the target master site for the SNAP1.WORLD
--snapshot site. The required snapshot logs need to be created at ORC1.WORLD.

CONNECT scott/tiger@orc1.world

CREATE SNAPSHOT LOG ON scott.emp;
CREATE SNAPSHOT LOG ON scott.dept;
CREATE SNAPSHOT LOG ON scott.bonus;
CREATE SNAPSHOT LOG ON scott.salgrade;


/*************************************************************************
STEP 2:
CREATE REPLICATED SCHEMA AND LINKS
*************************************************************************/

--Before you begin building your snapshot group, you must make sure that
--the replicated schema exists at the remote snapshot site and that the
--necessary database links have been created.

CONNECT system/manager@snap1.world

CREATE USER scott IDENTIFIED BY tiger;
GRANT connect, resource TO scott;

CONNECT scott/tiger@snap1.world

--The owner of the snapshots will need a database link pointing to the
--proxy_refresher that was created when the snapshot site was setup; see
--"CREATE MASTER SITE USERS" on page 2-6 for information.

CREATE DATABASE LINK orc1.world
   CONNECT TO proxy_refresher IDENTIFIED BY proxy_refresher;
```

```
/*************************************************************************
STEP 3:
CREATE SNAPSHOT GROUP
*************************************************************************/

--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--The master group that you specify in the GNAME parameter must match the
--name of the master group that you are replicating at the target master site.

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*************************************************************************
STEP 4:
CREATE REFRESH GROUP
*************************************************************************/

--All snapshots that are added to a particular refresh group will be
--refreshed at the same time. This ensures transactional consistency
--between the related snapshots in the refresh group.

BEGIN
   DBMS_REFRESH.MAKE (
      NAME => 'snapadmin.scott_rg',
      LIST => '',
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      IMPLICIT_DESTROY => FALSE,
      ROLLBACK_SEG => '',
      PUSH_DEFERRED_RPC => TRUE,
      REFRESH_AFTER_ERRORS => FALSE);
END;
/
```

```
/*************************************************************************
STEP 5:
ADD OBJECTS TO SNAPSHOT GROUP
*************************************************************************/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.bonus REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.bonus@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'dept',
      TYPE => 'SNAPSHOT',
      ddl_text => 'CREATE SNAPSHOT scott.dept REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.dept@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.emp REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.emp@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.salgrade REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.salgrade@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/

/*****************************************************************************
STEP 6:
ADD OBJECTS TO REFRESH GROUP
*****************************************************************************/

--Each of the snapshot group objects that you add to the refresh group
--will be refreshed at the same time to preserve referential integrity
--between related snapshots.

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.bonus',
      LAX => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.dept',
      LAX => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.emp',
      LAX => TRUE);
END;
/
```

```
BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.salgrade',
      LAX => TRUE);
END;
/
```

6

Conflict Resolution

This chapter illustrates how to define conflict resolution methods for your replicated environment. The following topics will be discussed:

- Prepare for Conflict Resolution
- Create Update Conflict Resolution Methods
- Create Uniqueness Conflict Resolution Methods
- Create Delete Conflict Avoidance Methods

Prepare for Conflict Resolution

Though you may take great care in designing your database and front-end application to avoid conflicts that may arise between multiple sites in a replicated environment, you may not be able to completely eliminate the possibility of conflicts. One of the most important aspects of replication is to ensure data convergence at all sites participating in the replicated environment.

When data conflicts do occur, you need a mechanism to ensure that the conflict will be resolved in accordance with your business rules and that the data converges correctly at all sites.

Oracle replication offers a variety of conflict resolution methods that will allow you to define a conflict resolution system for your database that will resolve conflicts in accordance with your business rules. If you have a unique situation that Oracle's pre-built conflict resolution methods cannot resolve, you have the option of building and using your own conflict routines.

Plan

Before you begin implementing conflict resolution routines for your replicated tables, you should take the time to analyze the data in your system to determine where the most conflicts may occur. For example, static data such as an employee number may change very infrequently and is not subject to a high occurrence of conflicts. An employee's customer assignments, however, may change often and would therefore be prone to data conflicts.

Once you have determined where the conflicts are most likely to occur, you need to determine how to resolve the conflict. Do you want the latest change to have precedence, or should one site over another have precedence?

As you read each of the sections describing the different conflict resolution routines, you will learn what each method is best suited for. Take the time to read each section and then think about how your business would want to resolve any potential conflicts.

After you have identified the potential "problem" areas and have determined what business rules would resolve the problem, use Oracle's conflict resolution routines (or one of your own) to implement a conflict resolution system.

Create Update Conflict Resolution Methods

The most common data conflict that you will encounter is when the same row at two (or more) different sites were updated at the same time (or before the deferred transaction from one site was successfully propagated to the other sites).

One method to resolve update conflicts is to implement a synchronous replicated environment, though this solution requires large network resource.

The other solution is to use the Oracle conflict resolution methods to deal with update conflicts that may occur when the same row has received two or more updates.

Overwrite and Discard

The overwrite and discard methods ignore the values from either the originating or destination site and therefore can never guarantee convergence with more than one master site. These methods are designed to be used by a single master site and multiple snapshot sites, or with some form of a user-defined notification facility.

The overwrite method replaces the current value at the destination site with the new value from the originating site. Conversely, the discard method ignores the new value from the originating site. See the "ADD_conflicttype_RESOLUTION procedure" section on page 8-85 and the "Overwrite and Discard" section in the *Oracle8i Replication* for more information.

> **Note:** This section will use objects not found in the other scripts within this book (since the configuration orc1.world, orc2.world, orc3.world, and snap1.world contains 3 master sites and 1 snapshot site and is not appropriate for OVERWRITE and DISCARD).

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@saturn.universe

--Before you can define any conflict resolution routines, you need to quiesce
--the master group that contains the table that you want to apply the conflict
--resolution routine to.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/
```

```
--All Oracle conflict resolution routines are based on logical column groupings
--termed "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      LIST_OF_COLUMN_NAMES => 'order,circumference,moons');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution routine for a specified table. This example will create a
--"Overwrite" conflict resolution routine.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'OVERWRITE',
      PARAMETER_COLUMN_NAME => 'order,circumference,moons');
END;
/

--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'titan',
      ONAME => 'planet',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/
```

Minimum and Maximum

When the advanced replication facility detects a conflict with a column group and calls either the *minimum* or *maximum* value conflict resolution methods, it compares the new value from the originating site with the current value from the destination site for a designated column in the column group. You must designate this column when you define your conflict resolution method.

If the new value of the designated column is *less than* or *greater than* (depending on the method used) the current value, the column group values from the originating site are applied at the destination site (assuming that all other errors were successfully resolved for the row), otherwise the rows will remain unchanged.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution routines, you need to quiesce
--the master group that contains the table that you want to apply the conflict
--resolution routine to.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
--All Oracle conflict resolution routines are based on logical column groupings
--termed "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      LIST_OF_COLUMN_NAMES => 'losal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution routine for a specified table. This example will create a
--"MINIMUM" conflict resolution routine.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'MINIMUM',
      PARAMETER_COLUMN_NAME => 'losal');
END;
/

--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Timestamp

The *earliest timestamp* and *latest timestamp* methods are variations on the minimum and maximum value methods. To use the timestamp method, you must designate a column in the replicated table of type DATE. When an application updates any column in a column group, the application must also update the value of the designated timestamp column with the local SYSDATE. For a change applied from another site, the timestamp value should be set to the timestamp value from the originating site.

There are several elements needed to make timestamp conflict resolution work well:

- Synchronized Time Settings Between Computers
- Timestamp field and trigger to automatically record timestamp

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution routines, you need to quiesce
--the master group that contains the table that you want to apply the conflict
--resolution routine to.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
--If the target table does not already contain a timestamp field,
--then you need to add an additional column to your table to record
--the timestamp value when a row is inserted or updated. Additionally,
--you must use the ALTER_MASTER_REPOBJECT API to apply the DDL to
--the target table (simply issuing the DDL may cause the replicated
--object to become invalid).

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (timestamp DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--you need to make sure that you re-generate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--Once the timestamp field has been created, you need to create a
--trigger that records the timestamp of when a row is either inserted
--or updated. This recorded value will be used in the resolution of
--conflicts based on the Timestamp method. Instead of directly executing the
--DDL, you should use the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure to
--create the trigger and add it to your master group.
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'TRIGGER',
      ONAME => 'insert_time',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE TRIGGER scott.insert_time
                      BEFORE
                         INSERT OR UPDATE ON scott.emp FOR EACH ROW
                      BEGIN
                         IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
                            :NEW.TIMESTAMP := SYSDATE;
                         END IF;
                      END;');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'insert_time',
      TYPE => 'TRIGGER',
      MIN_COMMUNICATION => TRUE);
END;
/

--All Oracle conflict resolution routines are based on logical column groupings
--termed "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, timestamp');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution routine for a specified table. This example will specify the
--"LATEST TIMESTAMP" conflict resolution routine using the TIMESTAMP column
--that you created earlier.
```

```
BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'LATEST TIMESTAMP',
      PARAMETER_COLUMN_NAME => 'timestamp');
END;
/

--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Additive and Average

The *additive* and *average* methods work with column groups consisting of a single numeric column only. Instead of "accepting" one value over another, this conflict resolution method either adds the two compared values together or takes an average of the two compared values.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world
```

```
--Before you can define any conflict resolution routines, you need to quiesce
--the master group that contains the table that you want to apply the conflict
--resolution routine to.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--All Oracle conflict resolution routines are based on logical column groupings
--termed "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      LIST_OF_COLUMN_NAMES => 'sal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution routine for a specified table. This example will specify the
--"ADDITIVE" conflict resolution routine using the SAL column.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'ADDITIVE',
      PARAMETER_COLUMN_NAME => 'sal');
END;
/
```

```
--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Priority Groups

Priority groups allow you to assign a priority level to each possible value of a particular column. If Oracle detects a conflict, Oracle updates the table whose "priority" column has a lower value using the data from the table with the higher priority value.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--You need to make sure that the JOB field is part of the column group that your
--site priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing column group.
--If you do not already have a column group, you can create a new column group
--using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.
```

```
BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, job');
END;
/

--Before you begin assigning a priority value to the values in your table, you
--must create a priority group that will "hold" the values that you defined.

BEGIN
   DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      DATATYPE => 'VARCHAR2');
END;
/

--The DBMS_REPCAT.ALTER_PRIORITY_datatype procedure is avaiable in several
--different versions; there is a version for each available datatype
--(NUMBER, VARCHAR2, etc.)  See "ALTER_PRIORITY_datatype procedure"
-- on page 8-94 for more information.  Execute this API as often as
--necessary until you have dfined a priority value for all possible
--table values.

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'president',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'manager',
      PRIORITY => 80);
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'salesman',
      PRIORITY => 60);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'analyst',
      PRIORITY => 40);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'clerk',
      PRIORITY => 20);
END;
/

--After you have completed assigning your priority values, you need to add the
--PRIORITY GROUP resolution method to your replicated table. The following API
--examples shows that it is the second conflict resolution method for the
--specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 2,
      METHOD => 'PRIORITY GROUP',
      PARAMETER_COLUMN_NAME => 'job',
      PRIORITY_GROUP => 'job_pg');
END;
/
```

```
--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Site Priority

Site priority is a specialized form of priority groups. Thus, many of the procedures associated with site priority behave similarly to the procedures associated with priority groups. Instead of resolving conflicts based on the priority of a field's value, the conflict will be resolved based on the priority of the sites involved.

For example, if you assign ORC2.WORLD a higher priority value than ORC1.WORLD and a conflict arises between these two sites, the value from ORC2.WORLD will be used.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--You will need to add a SITE column to your table to store the site value in
--your replicated table. Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure
--to apply the DDL to the target table (simply issuing the DDL may cause
--the replicated object to become invalid).
```

```
BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (site VARCHAR2(20))');
END;
/

--After you have inserted a new column into your replicated object,
--you need to make sure that you re-generate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After you have added the SITE column to your table, you need to make
--sure that this field is part of the column group that your site
--priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing
--column group. If you do not already have a column group, you can create a
--new column group using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, site');
END;
/

--Before you begin assigning a site priority value to the sites in your
--replicated environment, you must create a site priority group that will "hold"
--the values that you defined.
```

```
BEGIN
   DBMS_REPCAT.DEFINE_SITE_PRIORITY ((
      GNAME => 'scott_mg',
      NAME => 'site_pg');
END;
/

--Define the priority value for each of the sites in your replication
--environment using the DBMS_REPCAT.ADD_SITE_PRIORITY_SITE procedure.
--Execute this API as often as necessary until you have defined a site
--priority value for each of the sites in our replication environment.

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc1.world',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc2.world',
      PRIORITY => 50);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc3.world',
      PRIORITY => 25);
END;
/
```

```
--After you have completed assigning your site priority values, you need to
--add the SITE PRIORITY resolution method to your replicated table. The
--following API examples shows that it is the third conflict resolution method
--for the specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 3,
      METHOD => 'site priority',
      PARAMETER_COLUMN_NAME => 'site',
      PRIORITY_GROUP => 'site_pg');
END;
/

--After you have defined your conflict resolution routine, you need to
--regenerate replication support for the table that received the conflict
--resolution routine.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Uniqueness Conflict Resolution Methods

In a replicated environment, you may encounter situations where you will receive a conflict on a unique constraint, often resulting from an insert. If your business rules allow you to delete the duplicate row, you can define such resolution with Oracle's pre-built conflict resolution routines.

More often, however, you will want to modify the conflicting value so that it no longer violates the unique constraint; modifying the conflicting value will ensure that you don't lose important data. Oracle's pre-built uniqueness conflict resolution routine can make the conflicting value unique by appending a site name or a sequence number to the value.

An additional component that accompanies uniqueness conflict routines is a notification facility. The conflicting information will be modified by Oracle so that it can be inserted into the table, but you should be notified so that you can analyze the conflict to determine if the record should be deleted, or the data merged into another record, or a completely new value be defined for the conflicting data.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--As you might expect, a uniqueness conflict resolution routine detects and
--resolves conflicts encounted on a column(s) with a UNIQUE constraint. Use
--the ALTER_MASTER_REPOBJECT procedure (described on page 8-91) to add
--a UNIQUE constraint to the EMP table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD
                  (constraint emp_ename_unique UNIQUE(ename))');
END;
/
```

```
--After you have add the UNIQUE constraint to your replicated table,
--you need to make sure that you re-generate replication support for
--the affected table. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--The following table (conf_report) will store the messages received from
--your notification facility.

BEGIN
   DBMS_REPCAT.EXECUTE_DDL(
      GNAME => 'scott_mg',
      DDL_TEXT => 'CREATE TABLE scott.conf_report (
                   line NUMBER(2),
                   txt VARCHAR2(80),
                   timestamp DATE,
                   table_name VARCHAR2(30),
                   table_owner VARCHAR2(30),
                   conflict_type VARCHAR2(7))');
END;
/

CONNECT scott/tiger@orc1.world

--The following package (notify) will send a notification to the CONF_REPORT
--table when a conflict is detected.

--The conflict resolution notification package that is created in this script is
--described in detail in the Oracle8i Replication book (see the "User-Defined
--Conflict Notification Methods" section to learn more about the following
--package and procedures).
```

```
CREATE OR REPLACE PACKAGE notify AS
   FUNCTION emp_unique_violation(ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN;
END notify;
/

CREATE OR REPLACE PACKAGE BODY notify AS
   TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;
   PROCEDURE report_conflict(conflict_report IN MESSAGE_TABLE,
      report_length IN NUMBER,
      conflict_time IN DATE,
      conflict_table IN VARCHAR2,
      table_owner IN VARCHAR2,
      conflict_type IN VARCHAR2) IS
      BEGIN
         FOR idx IN 1..report_length LOOP
            BEGIN
               INSERT INTO scott.conf_report
               (line, txt, timestamp, table_name, table_owner, conflict_type)
               VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
               conflict_table, table_owner, conflict_type);
            EXCEPTION WHEN others THEN NULL;
            END;
         END LOOP;
      END report_conflict;
   FUNCTION emp_unique_violation(ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN IS
      local_node VARCHAR2(128);
      conf_report MESSAGE_TABLE;
      conf_time DATE := SYSDATE;
   BEGIN
      BEGIN
         SELECT global_name INTO local_node FROM global_name;
      EXCEPTION WHEN others THEN local_node := '?';
      END;
      conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE EMP ON ' ||
         TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
      conf_report(2) := ' AT NODE ' || local_node;
      conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING' ||
         ' APPEND SITE NAME METHOD';
      conf_report(4) := 'ENAME: ' || ename;
      conf_report(5) := NULL;
      report_conflict(conf_report, 5, conf_time, 'EMP', 'SCOTT', 'UNIQUE');
```

```
      discard_new_values := FALSE;
      RETURN FALSE;
   END emp_unique_violation;
END notify;
/

CONNECT repadmin/repadmin@orc1.world

--The following package will be replicated to all of the master sites in your
--replication environment; this will ensure that the notification facility is
--available at all master sites.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE BODY',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

--After you have completed building your notification facility you need to
--add the notification facility as one of your conflict resolution methods
--(even though it only notifies of a conflict). The following API example will
--demonstrate adding the notification facility as a USER FUNCTION.
```

```
BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 1,
      METHOD => 'USER FUNCTION',
      COMMENT => 'Notify DBA',
      PARAMETER_COLUMN_NAME => 'ename',
      FUNCTION_NAME => 'scott.notify.emp_unique_violation');
END;
/

--After you have added the notification facility, you are ready to add the
--actual conflict resolution method to your table. The following API example
--will demonstrate adding the APPEND SITE NAME uniqueness conflict resolution
--routine to your replicated table.

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 2,
      METHOD => 'APPEND SITE NAME',
      PARAMETER_COLUMN_NAME => 'ename');
END;
/

--After you have defined your conflict resolution routine(s), you need to
--regenerate replication support for the table that received the conflict
--resolution routine(s).

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.
```

```
BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Delete Conflict Avoidance Methods

Unlike update conflicts where there are two values to compare, simply deleting a row makes the update conflict resolution methods described in the previous section ineffective since only one value would exist.

The best way to deal with deleting rows in a replication environment is to "avoid" the conflict by marking a row for deletion and periodically purging the table of all "marked" records. Since you are not physically removing this row, your data will be able to converge at all master sites if a conflict arises because you still have two values to compare (assuming that no other errors have occurred). After you are sure that your data has converged, you can purge "marked" rows using a replicated purge procedure.

When you are developing your front-end application for your database, you will probably want to "filter out" the columns that have been marked for deletion; this will make it appear to your users as though the row was physically deleted. Simply exclude the rows that have been marked for deletion in the SELECT statement for your data set; for example, a select statement for a current employee listing might look like:

```
SELECT * FROM emp WHERE remove_date IS NULL;
```

This section will describe how to prepare your replicated table to avoid delete conflicts. You will also see how to use procedural replication to purge those records that have been "marked" for deletion.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
--You need to add a column to your replicated table that will store the
--mark for deleted records. It is advisable to use a timestamp to mark your
--records for deletion (timestamp will reflect when the record was marked for
--deletion. Since you will use a timestamp, your new column will need to be
--a DATE datatype.  Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure to add
--the REMOVE_DATE column to your existing replicated table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (remove_date DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--you need to make sure that you re-generate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--The following package will be replicated to all of the master sites in your
--replication environment. This package will purge all "marked" records from
--the specified table.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE scott.purge AS
                      PROCEDURE remove_emp(purge_date DATE);
                   END;');
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE BODY',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE BODY scott.purge AS
                      PROCEDURE remove_emp(purge_date IN DATE) IS
                      BEGIN
                         DBMS_REPUTIL.REPLICATION_OFF;
                         LOCK TABLE scott.emp IN EXCLUSIVE MODE;
                         DELETE scott.emp WHERE remove_date IS NOT NULL AND
                            remove_date < purge_date;
                         DBMS_REPUTIL.REPLICATION_ON;
                      EXCEPTION WHEN others THEN
                         DBMS_REPUTIL.REPLICATION_ON;
                      END;
                   END;');
END;
/

--After you have created your package (package and package body), you need to
--generate replication support for each component. After you generate
--replication support, a synonym will be created for you and added to your
--master group as a replicated object. This synonym will be labeled as
--DEFER_PURGE.REMOVE_EMP.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE BODY',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, you need to resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

7

Manage Replicated Environment with APIs

This chapter illustrates how to manage your replication environment using the Replication API set. The following topics will be discussed:

- Managing Master Sites
- Managing Snapshot Sites
- Managing the Error Queue
- Alter Replicated Object
- Offline Instantiation

Managing Master Sites

As your data delivery needs change due to growth, shrinkage, or emergencies, you are undoubtedly going to need to change the configuration of your replication environment. This section is devoted to managing the master sites of your replication environment, which will help you alter and reconfigure your master sites.

Change Master Definition Site

Many replication administrative tasks can only be performed from the master definition site. Use the DBMS_REPCAT.RELOCATE_MASTERDEF procedure to move the master definition site to another master site. This API is especially useful when the master definition site becomes unavailable and you need to specify a new master definition site (see "Option 2" below).

Option 1

If all master sites are available, complete the following:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      GNAME => 'scott_mg',
      OLD_MASTERDEF => 'orc1.world',
      NEW_MASTERDEF => 'orc2.world',
      NOTIFY_MASTERS => TRUE,
      INCLUDE_OLD_MASTERDEF => TRUE);
END;
/
```

Option 2

If the old master definition site is NOT available, complete the following:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      GNAME => 'scott_mg',
      OLD_MASTERDEF => 'orc1.world',
      NEW_MASTERDEF => 'orc2.world',
      NOTIFY_MASTERS => TRUE,
      INCLUDE_OLD_MASTERDEF => FALSE);
END;
/
```

See "RELOCATE_MASTERDEF procedure" on page 8-151 for more information on using this procedure.

Add a Master Site

As your replicated environment expands, you will need to use the ADD_MASTER_DATABASE procedure to add additional master sites to an existing master group. Executing this procedure will replicate existing master object to the new site.

Before you add a new master site, be sure that you properly setup your new master site for replication. Make sure that you follow the steps described in the "Setup Master Site" section on page 2-4.

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc4.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repsites WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Drop a Master Site

When it becomes necessary to remove a master site from a master group, use the REMOVE_MASTER_DATABASES procedure API to drop one or more master sites.

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
BEGIN
   DBMS_REPCAT.REMOVE_MASTER_DATABASES (
      GNAME => 'scott_mg',
      MASTER_LIST => 'orc4.world');
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Managing Snapshot Sites

Snapshot replication provides you with the flexibility to build data sets to meet the needs of your users, security configuration, and front-end applications. The following two sections will describe how you can create multiple data sets of the same target master group at a single snapshot site. You will also learn how to manually push your snapshot's deferred transaction queue.

Using a Group Owner

Specifying a group owner when you define a new snapshot group and its related objects allows you to create multiple snapshot groups based on the same master group at a single snapshot site. See "Organizational Mechanisms" in Chapter 3 of the *Oracle8i Replication* manual for a complete discussion on using group owners and the advantages of using multiple data sets.

```
--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--The master group that you specify in the GNAME parameter must match the
--name of the master group that you are replicating at the target master site.
--The GOWNER parameter allows you to specify an additional identifier that lets
--you create multiple snapshot groups based on the same master group at the same
--snapshot site.
```

```
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS',
      GOWNER => 'bob');
END;
/

--The GOWNER value used when creating your snapshot obects must match the GOWNER
--value specified when you created the snapshot group (previous procedure). In
--this example, you will specify BOB as the group owner to add these objects to
--the SCOTT_MG snapshot group owned by BOB.

--WARNING: You need to make sure that each object created has a unique name.
--When using a GOWNER to create multiple snapshot groups, duplicate object names
--could become a problem. To avoid any object naming conflicts, you may want to
--append the GOWNER value to the end of the object name that you are create, as
--illustrated in the following procedures (i.e. CREATE SNAPSHOT scott.bonus_bob);
--such a naming method will ensure that you do not create any objects with
--conflicting names.

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'bonus_bob',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.bonus_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.bonus@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'dept_bob',
      TYPE => 'SNAPSHOT',
      ddl_text => 'CREATE SNAPSHOT scott.dept_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.dept@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'emp_bob',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.emp_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.emp@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'salgrade_bob',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.salgrade_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.salgrade@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

--After you have completed building your snapshot group, you should add your
--snapshots to a refresh group. See Chapter 5, "Create a Snapshot Group"
--(step 6) for more information about adding snapshots to a refresh group.
```

Pushing the Deferred Transaction Queue

If you do not automatically propagate the transactions in your deferred transaction queue during the refresh of your snapshot, you will need to complete the following steps to propagate changes made to the updateable snapshot to its master table.

```
--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--Propagation of the deferred transaction queue is based on the destination of
--the transaction. Execute the following SELECT statement to view the deferred
--transactions and their destinations (each distinct destination and the number
--of transactions pending for the destination wil be displayed):

SELECT DISTINCT(dblink), COUNT(deferred_tran_id)
   FROM deftrandest GROUP BY dblink;

--You will need to execute the DBMS_DEFER_SYS.PUSH function for each master
--site that is listed as a destination for a deferred transaction.

DECLARE
   temp INTEGER;
BEGIN
   temp := DBMS_DEFER_SYS.PUSH (
      DESTINATION => 'orc1.world',
      STOP_ON_ERROR => FALSE,
      DELAY_SECONDS => 0,
      PARALLELISM => 0);
END;
/

--Repeat the above procedure for each destination that was returned in the above
--SELECT statement.
```

Dropping Snapshot Sites

There may be many different reasons why you need to drop replication activity at a snapshot site. Perhaps the data requirements have changed or an employee has left the company. In any case, as a DBA you will need to drop the replication support for the target snapshot site.

Drop Snapshot Group Created with Deployment Templates

The process for dropping a snapshot group that was created by instantiating a deployment template at a snapshot site is slightly different than the following methods described in the next couple of sections. Before you drop the snapshot group at the remote snapshot site, you need to execute the DROP_SITE_INSTANTIATION procedure at the target master site for snapshot group. In addition to removing the meta data relating to the snapshot group, this procedure will also remove the related deployment template data regarding this site.

There is a public and a private version of the DROP_SITE_INSTANTIATION procedure. The public version allows the owner of the snapshot group to drop the snapshot site, while the private version allows the replication administrator to drop a snapshot site on behalf of the snapshot group owner.

Public The following steps are to be performed by owner of the snapshot group.

Executed As: Snapshot Group Owner

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT scott/tiger@orc1.world

--If you need to drop a snapshot site that was instantiated on an Oracle8i Lite
--database, see the Oracle8i Lite documentation for information.

DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
   REFRESH_TEMPLATE_NAME =>  'personnel',
   SITE_NAME => 'snap1.world');
/

--After you have executed the DROP_SITE_INSTANTIATION procedure, you should
--connect to the remote snapshot site and drop the snapshot group (if you are
--not able to connect to the remote snapshot site due to loss or theft, the
--target snapshot group will not be able to refresh, but the existing data will
--still remain at the snapshot site).
```

```
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   GNAME => 'scott_mg',
   DROP_CONTENTS => TRUE);
/
```

Private The following steps are to be performed by the replication administrator on behalf of the snapshot group owner.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

--If you need to drop a snapshot site that was instantiated on an Oracle8i Lite
--database, see the Oracle8i Lite documentation for information.

DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
   REFRESH_TEMPLATE_NAME =>  'personnel',
   USER_NAME => 'scott',
   SITE_NAME => 'snap1.world');
/

--After you have executed the DROP_SITE_INSTANTIATION procedure, you should
--connect to the remote snapshot site and drop the snapshot group (if you are
--not able to connect to the remote snapshot site due to loss or theft, the
--target snapshot group will not be able to refresh, but the existing data will
--still remain at the snapshot site).

CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   GNAME => 'scott_mg',
   DROP_CONTENTS => TRUE);
/
```

Drop Snapshot Objects at Snapshot Site

The most secure method of removing replication support for a snapshot site is to physically drop the replicated objects and/or groups at the snapshot site. The following two sections will describe how to drop these objects and groups while connected to the snapshot group.

Ideally, these procedures should be executed while the snapshot is connected to its target master site; a connection will ensure that any related metadata at the master site is removed. If a connection to the master site is not possible, be sure to complete the procedure described in the "Cleanup Master Site" on page 7-12 to manually remove the related metadata.

Drop Snapshot Group at Snapshot Site When it becomes necessary to remove a snapshot group from a snapshot site, use the DROP_SNAPSHOT_REPGROUP procedure to drop a snapshot group. When you execute this procedure and are connected to the target master site, the meta data for the target snapshot group at the master site will be removed (if you are not able to be connected, see "Cleanup Master Site" on page 7-12 for more information).

Executed As: Snapshot Administrator

Executed At: Remote Snapshot Site

Replication Status: N/A

```
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   GNAME => 'scott_mg',
   DROP_CONTENTS => TRUE);
/
```

Drop Individual Snapshot at Snapshot Site When it becomes necessary to remove an individual snapshot from a snapshot site, use the DROP_SNAPSHOT_REPOBJECT procedure API to drop a snapshot. When you execute this procedure, the meta data for the target snapshot at the master site will be removed. When you execute this procedure and are connected to the target master site, the meta data for the target snapshot group at the master site will be removed (if you are not able to be connected, see "Cleanup Master Site" on page 7-12 for more information).

Executed As: Snapshot Administrator

Executed At: Remote Snapshot Site

Replication Status: N / A

```
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT (
   SNAME => 'scott',
   ONAME => 'bonus',
   TYPE => 'SNAPSHOT',
   DROP_OBJECTS => TRUE);
/
```

Cleanup Master Site

If you are unable to drop snapshot group or snapshot object while connected to the target master site, you will need to manually remove the related metadata at the master site. Cleaning up the metadata will also ensure that you are not needlessly maintaining master table changes to a snapshot log. The following sections will help you cleanup your master site after dropping a snapshot group or object.

Cleanup After Dropping Snapshot Group If you have executed the steps described in the "Drop Snapshot Group at Snapshot Site" section on page 7-11 and were not connected to the master site, you are encouraged to complete the following steps to cleanup the target master site.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP (
   GNAME => 'scott_mg',
   SNAPSITE => 'snap1.world');
/
```

```
--After you unregister the snapshot group, you should purge the snapshot logs
--of the entries that were marked for the target snapshots. The
--PURGE_SNAPSHOT_FROM_LOG procedure will need to be executed for each snapshot
--that was in the snapshot replication group.

--NOTE: If for some reason unregistering the snapshot group fails, you are still
--encouraged to complete the following steps.

DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   SNAPOWNER => 'scott',
   SNAPNAME => 'emp',
   SNAPSITE => 'snap1.world');
/

DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   SNAPOWNER => 'scott',
   SNAPNAME => 'dept',
   SNAPSITE => 'snap1.world');
/

DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   SNAPOWNER => 'scott',
   SNAPNAME => 'bonus',
   SNAPSITE => 'snap1.world');
/

DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   SNAPOWNER => 'scott',
   SNAPNAME => 'salgrade',
   SNAPSITE => 'snap1.world');
/
```

Cleanup Individual Snapshot Support at Master Site If you have executed the steps described in the "Drop Individual Snapshot at Snapshot Site" section on page 7-11 and were not connected to the master site, you are encouraged to complete the following steps to cleanup the target master site.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world
```

```
DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT (
   SNAPOWNER => 'scott',
   SNAPNAME => 'bonus',
   SNAPSITE => 'snap1.world');
/

--After you unregister the snapshot, you should purge the associated snapshot
--log of the entries that were marked for the target snapshots.

--NOTE: If for some reason unregistering the snapshot fails, you are still
--encouraged to complete the following step.

DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   SNAPOWNER => 'scott',
   SNAPNAME => 'bonus',
   SNAPSITE => 'snap1.world');
/
```

Managing the Error Queue

As an administrator of a replication environment, you should regularly monitor the error queue to determine if any deferred transactions were not successfully applied at the target master site.

To check the error queue, issue the following SELECT statement as the replication administrator when connected to the target master site:

```
SELECT * FROM deferror;
```

If the error queue contains errors, you should resolve the error condition and re-execute the deferred transaction. You have two options when re-executing a deferred transaction: you can re-execute in the security context of the user who received the deferred transaction or you can re-execute the deferred transaction with an alternate security context.

Re-execute Error Transaction as the Receiver

The procedure below will re-execute a specified deferred transaction in the security context of the user that received the deferred transaction. This procedure should not be executed until the error situation has been resolved.

Executed As: Replication Administrator

Executed At: Site Containing Errors

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc2.world');
END;
/
```

Re-execute Error Transaction as Alternate User

The procedure below will re-execute a specified deferred transaction in the security context of the currently connected user. This procedure should not be executed until the error situation has been resolved.

Executed As: Connected User

Executed At: Site Containing Errors

Replication Status: Normal

```
CONNECT scott/tiger@orc2.world

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc2.world');
END;
/
```

Alter Replicated Object

As your database needs change, you may need to modify the characteristics of your replicated objects. It is important that you do not directly execute DDL to alter your replicated objects; doing so may cause your replicated environment to fail.

Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure to alter the characteristics of your replicated objects. From the example below, you will see that you simply include the necessary DDL within the procedure call (see the DDL_TEXT parameter).

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (site VARCHAR2(20))');
END;
/

--After you have inserted a new column into your replicated object,
--you need to make sure that you re-generate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Offline Instantiation

Expanding established replicated environments can cause extreme network traffic when you add a new master or snapshot site to your replicated environment. This is caused by propagating the entire contents of the table or snapshot via the network to the new replicated site.

To alleviate such network traffic, you can expand your replicated environment by using the offline instantiation procedure. Offline instantiation takes advantage of Oracle's export and import utilities, which allows you to create an export file and transfer the data to the new site via another storage media (i.e. CD-ROM, tape, etc.).

Master Site

The following script is an example of how to perform an offline instantiation of a master site. This script can potentially save large amounts of network traffic cause by the normal method of adding a new master site to an existing master group.

Executed As: Replication Administrator

Executed At: Master Definition Site and New Master Site

Replication Status: Quiesced and Partial

```
/*************************************************************************
SETUP NEW MASTER SITE

You need to complete the steps illustrated in the
"Setup Master Site" section on page 2-4. You will need
to make sure that the appropriate schema and database links have been
created before you perform the offline instantiate of your new master
site. Be sure to create the database links from the new master site to
each of the existing masters sites; you will also need to create a
database link from each of the existing master sites to the new
master site.

After the database links have been created, make sure that you also
define the SCHEDULED LINKS for each of the new database links (STEP 8:
CREATE SCHEDULED LINKS).
*************************************************************************/

/*************************************************************************
SUSPEND MASTER ACTIVITY

You need to suspend master activity for the existing master sites
before you export your master data and begin the offline instantiation
process.
*************************************************************************/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

/*************************************************************************
VERIFY THAT THERE ARE NO PENDING TRANSACTIONS

This includes that you push any outstanding deferred transactions,
resolve any error transactions, and/or push any administrative
transactions. This step needs to be performed at each of the existing
master sites.
*************************************************************************/

--connect to master definition site

CONNECT repadmin/repadmin@orc1.world
```

```
--Check for error transaction queue.

SELECT * FROM deferror;

--If any deferred transactions have been entered into the error queue, then
--you need to resolve the error situation and then manually re-execute the
--deferred transaction.

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc1.world');
END;
/

--Check for outstanding administrative requests.

SELECT * FROM dba_repcatlog;

--If any administrative requests remain, then you can manually push these
--transactions and/or wait for them to be executed automatically. You may need
--to execute the DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN API several times, since
--some administrative operations have multiple steps.

BEGIN
   DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
      GNAME => 'scott_mg',
      ALL_SITES => TRUE);
END;
/

/*************************************************************************
BEGIN OFFLINE INSTANTIATION PROCEDURE
*************************************************************************/
--Connect as replication administrator to Master Definition Site

connect repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

```
/*
NOTE: You must wait until the DBA_REPCATLOG view is empty before
continuing the steps in this script. Execute the following SELECT statement
in another SQL*Plus session to monitor your DBA_REPCATLOG view:

SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';
*/
PAUSE Press <RETURN> to continue.


/*********************************************************************
CONNECT AS SCOTT/TIGER TO EXPORT

You will need to use the Oracle export utility to generate the export
file that you will transfer to the new master site. The export file
will contain the replicated objects to be added at the new master site.
See the Oracle8i Utilities book for additional information.
*********************************************************************/

EXP80 scott/tiger@orc1.world

/*********************************************************************
RESUME PARTIAL REPLICATION ACTIVITY

Since it may take you some time to complete the offline instantiation
process, you can resume replication activity for the existing
master sites by executing the DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS
procedure after the export is complete.
*********************************************************************/

--connect as replication administrator to master definition site.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

```
/*****************************************************************
PREPARE NEW MASTER SITE

After you have transferred the export file from the master definition
site to the new master site, you must prepare the new site to import
the data in your export file.
*****************************************************************/

CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_OFFLINE_OG.BEGIN_LOAD (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/

/*****************************************************************
IMPORT DATA FROM EXPORT FILE

Once you have imported the export file that you generated earlier,
you will have transferred the data from your master definition site
to your new master site.
*****************************************************************/
IMP80 scott/tiger@orc4.world FULL=y IGNORE=y

/*****************************************************************
COMLETE LOAD PROCESS AT NEW MASTER SITE

After you have imported the export file, you are ready to complete
the offline instantiation process at the new master site. Executing
DBMS_OFFLINE_OG.END_LOAD procedure will prepare the new site for
normal replication activity.
*****************************************************************/

CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_OFFLINE_OG.END_LOAD (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

```
/*****************************************************************
COMPLETE INSTANTIATION PROCESS

Once you have complete the steps at the new master site, you are ready
to complete the offline instantiation process. Executing the
DBMS_OFFLINE_OG.END_INSTANTIATION procedure will complete the process
and resume normal replication activity at all master sites.
*****************************************************************/
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.END_INSTANTIATION (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

Snapshot Site

For the same reasons that you might want to perform an offline instantiation of a master site, you may also want to create a new snapshot group at a snapshot site using the offline instantiation process. In some cases, it is even more useful for snapshots considering that the target computer could very well be a laptop using a modem connection.

The following script describes the process of performing an offline instantiation for a new snapshot group.

Executed As: Replication Administrator and Snapshot Administrator

Executed At: Master Site and New Snapshot Site

Replication Status: Normal

```
/*****************************************************************
SETUP SNAPSHOT SITE

You need to complete the steps illustrated in the
"Setup Snapshot Site" section on page 2-15. You will need
to make sure that the appropriate schema and database links have been
created before you perform the offline instantiation of your snapshot.
*****************************************************************/
```

```
/*****************************************************************************
CREATE SNAPSHOT LOGS

If snapshot logs do not already exist for the target master tables, you
will need to create them at the target master site.
*****************************************************************************/

CONNECT repadmin/repadmin@orc1.world

CREATE SNAPSHOT LOG ON scott.emp;
CREATE SNAPSHOT LOG ON scott.dept;
CREATE SNAPSHOT LOG ON scott.bonus;
CREATE SNAPSHOT LOG ON scott.salgrade;

/*****************************************************************************
CREATE TEMPORARY SNAPSHOTS

You will create temporary snapshots at the master site that will contain the
data that you will transfer to your new snapshot site using the export file.

NOTE: If you added any of the conflict resolution routines described in
Chapter 6, "Conflict Resolution", you may have additional columns in your
tables. Be certain to include these additional columns in the SELECT
statements below; updatable snapshots require that you explicity select all
columns in the master table (no SELECT *).
*****************************************************************************/

CREATE SNAPSHOT scott.snap_emp REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
   FROM scott.emp@orc1.world;

CREATE SNAPSHOT scott.snap_dept REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT deptno, dname, loc
   FROM scott.dept@orc1.world;

CREATE SNAPSHOT scott.snap_bonus REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT ename, job, sal, comm
   FROM scott.bonus@orc1.world;

CREATE SNAPSHOT scott.snap_salgrade REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT grade, losal, hisal
   FROM scott.salgrade@orc1.world;
```

```
/*****************************************************************
CONNECT AS SCOTT/TIGER TO EXPORT

You will need to use the Oracle export utility to generate the export
file that you will transfer to the new snapshot site. The export file
will contain the base tables of your temporary snapshots. See the
Oracle8i Utilities book for additional information.

NOTE: The following example is to be used for Oracle8i databases
only. Base tables in database versions earlier than Oracle8i will
be preceded by the SNAP$ prefix (i.e. SNAP$_SNAP_EMP).
*****************************************************************/

EXP80 scott/tiger@orc1.world TABLES='snap_emp','snap_dept',
'snap_bonus','snap_salgrade'

/*****************************************************************
DELETE THE TEMPORARY SNAPSHOTS

After you have completed your export, you should delete the temporary
snapshots that you created during the beginning of this procedure.
*****************************************************************/

CONNECT scott/tiger@orc1.world

DROP SNAPSHOT snap_emp;
DROP SNAPSHOT snap_dept;
DROP SNAPSHOT snap_bonus;
DROP SNAPSHOT snap_salgrade;

/*****************************************************************
CREATE NECESSARY SCHEMA AND DATABASE LINK

Before you perform the offline instantiation of your snapshots, you
need to create the schema that will contain the snapshots at the new
snapshot site (they need to be in the same schema that contains the
master objects at the master site) and the database link from the
snapshot site to the master site.
*****************************************************************/
CONNECT system/manager@snap2.world

CREATE USER scott IDENTIFIED by tiger;

GRANT connect, resource TO scott;
```

```
CONNECT scott/tiger@snap2.world

CREATE DATABASE LINK orc1.world CONNECT TO scott IDENTIFIED by tiger;

/*************************************************************************
CREATE EMPTY SNAPSHOT GROUP

You need to execute the DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP API at
the new snapshot site to contain an empty snapshot group that you
will add your snapshots to.
*************************************************************************/

CONNECT snapadmin/snapadmin@snap2.world

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*************************************************************************
PREPARE SNAPSHOT SITE FOR OFFLINE INSTANTIATION

The DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD API creates the necessary support
mechanisms for the new snapshots. This step also adds the new
snapshots to the snapshot group that you created in the previous step.

Be sure to execute the DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD API for each
snapshot that you will be importing.
*************************************************************************/

CONNECT system/manager@snap2.world

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_emp');
END;
/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_dept');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_bonus');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_salgrade');
END;
/

/*********************************************************************
CONNECT AS SCOTT/TIGER TO IMPORT AT NEW SNAPSHOT SITE

You will need to use the Oracle import utility to import the file
that you exported earlier. Make sure that you import your data as the
same user that exported the data (i.e. scott/tiger).
*********************************************************************/

IMP80 scott/tiger@snap2.world FULL=y IGNORE=y

/*********************************************************************
COMPLETE THE OFFLINE INSTANTIATION

Execute the DBMS_OFFLINE_SNAPSHOT.END_LOAD API to finish the offline
instantiation of the imported snapshots.
*********************************************************************/

CONNECT system/manager@snap2.world
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_emp');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_dept');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_bonus');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_salgrade');
END;
/

/*****************************************************************
REFRESH SNAPSHOTS TO REGISTER AT MASTER SITE

In addition to retreiving the latest changes from the master tables,
refreshing the snapshots at the new snapshot site registers the offline
instantiated snapshots at the target master site.
*****************************************************************/

CONNECT scott/tiger@snap2.world
```

```
BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_emp');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_dept');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_bonus');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_salgrade');
END;
/
```

8

Replication Management API Reference

All installations of Oracle advanced replication include the replication management application programming interface (API). A server's *replication management API* is a set of PL/SQL packages that encapsulates procedures and functions that administrators can use to configure Oracle's advanced replication features. Oracle Replication Manager also uses the procedures and functions of each site's replication management API to perform work. This chapter describes that packages that constitute Oracle replication API, including:

- The procedures and functions in each package.
- The parameters for each packaged procedure or function.
- Exceptions that each procedure or function can raise.

Packages

Oracle's replication management API includes the following packages:

- DBMS_DEFER Package
- DBMS_DEFER_QUERY Package
- DBMS_DEFER_SYS Package
- DBMS_OFFLINE_OG Package
- DBMS_OFFLINE_SNAPSHOT Package
- DBMS_RECTIFIER_DIFF Package
- DBMS_REFRESH Package
- DBMS_REPCAT Package
- DBMS_REPCAT_ADMIN Package
- DBMS_REPCAT_INSTANTIATE Package
- DBMS_REPCAT_RGT Package
- DBMS_REPUTIL Package
- DBMS_SNAPSHOT Package

Examples of Using Oracle's Replication Management API

To use Oracle's replication management API, you issue procedure or function calls using an ad-hoc query tool such as an Enterprise Manager SQL Worksheet, Server Manager's command prompt, or SQL*Plus. For example, the following call to the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure creates a new replicated table SALES.EMP in the ACCT replication group.

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     sname                => 'sales',
     oname                => 'emp',
     type                 => 'table',
     use_existing_object  => TRUE,
     ddl_text             => 'CREATE TABLE acct_rec.emp AS . . .',
     comment              => 'created by . . .',
     retry                => FALSE,
     copy_rows            => TRUE,
     gname                => 'acct');
```

To call a replication management API function, you must provide an environment to receive the return value of the function. For example, the following anonymous PL/SQL block calls the DBMS_DEFER_SYS.DISABLED function in an IF statement.

```
BEGIN
 IF DBMS_DEFER_SYS.DISABLED('inst2') THEN
  DBMS_OUTPUT.PUT_LINE('Propagation to INST2 is disabled.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Propagation to INST2 is enabled.');
 END IF;
END;
```

Prerequisites to Consider

For many procedures and functions in the replication management API, there are important prerequisites to consider. For example:

- Some procedures or functions are appropriate to call only from the master definition site in a multimaster configuration.
- To perform certain administrative operations for master groups, you must first suspend replication activity for the group before calling replication management API procedures and functions.
- The order in which you call different procedures and functions in Oracle's replication management API is extremely important. See the next section for more information about learning how to correctly issue replication management calls.

Replication Manager and Oracle Replication Management API

Oracle's Replication Manager uses the replication management API to perform most of its functions. Using Replication Manager is much more convenient than issuing replication management API calls individually because the utility:

- Provides a GUI interface to type in and adjust API call parameters.
- Automatically orders numerous, related API calls in the proper sequence.
- Displays output returned from API calls in message boxes and error files.

An easy way to learn how to use Oracle's replication management API is to use Replication Manager scripting feature. When you start an administrative session with Replication Manager, turn scripting on. When you are finished, turn scripting off and then review the script file. The script file contains all replication management API calls that were made during the session. See the Replication Manager help documentation for more information about its scripting feature.

DBMS_DEFER Package

Summary of Subprograms

Table 8–1

Subprogram	Description
CALL procedure on page 8-6	Builds a deferred call to a remote procedure
COMMIT_WORK procedure on page 8-8	Performs a transaction commit after checking for well-formed deferred remote procedure calls
datatype_ARG procedure on page 8-9	Provides the data that is to be passed to a deferred remote procedure call
TRANSACTION procedure on page 8-10	Indicates the start of a new deferred transaction

CALL procedure

This procedure builds a deferred call to a remote procedure.

Syntax

```
DBMS_DEFER.CALL (
   schema_name       IN   VARCHAR2,
   package_name      IN   VARCHAR2,
   proc_name         IN   VARCHAR2,
   arg_count         IN   NATURAL,
   { nodes           IN   node_list_t
   | group_name      IN   VARCHAR2 :=''});
```

Parameters

Table 8–2 CALL Procedure Parameters

Parameter	Description
schema_name	Name of the schema in which the stored procedure is located.
package_name	Name of the package containing the stored procedure. The stored procedure must be part of a package. Deferred calls to standalone procedures are not supported.
proc_name	Name of the remote procedure to which you want to defer a call.
arg_count	Number of parameters for the procedure. You must have one call to DBMS_DEFER.datatype_ARG for each of these parameters.
nodes	A PL/SQL table of fully qualified database names to which you want to propagate the deferred call. The table is indexed starting at position 1 and ending when a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive. This argument is optional.
group_name	Reserved for internal use.

Note: The CALL procedure is overloaded. The nodes and group_name parameters are mutually exclusive.

Exceptions

Table 8–3 CALL Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous call was not correctly formed.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Destination list (specified by nodes or by a previous DBMS_DEFER.TRANSACTION call) contains duplicates.

COMMIT_WORK procedure

This procedure performs a transaction commit after checking for well-formed deferred remote procedure calls.

Syntax

```
DBMS_DEFER.COMMIT_WORK (
   commit_work_comment IN VARCHAR2);
```

Parameters

Table 8–4 COMMIT_WORK Procedure Parameters

Parameter	Description
commit_work_comment	Equivalent to SQL "COMMIT COMMENT" statement.

Exceptions

Table 8–5 COMMIT_WORK Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Transaction was not correctly formed or terminated.

*datatype*_ARG procedure

This procedure provides the data that is to be passed to a deferred remote procedure call. Depending upon the type of the data that you need to pass to a procedure, you must call one of the following procedures for each argument to the procedure.

Syntax

```
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);
```

Parameters

Table 8–6 datatype_ARG Procedure Parameters

Parameter	Description
arg	Value of the parameter that you want to pass to the remote procedure to which you previously deferred a call.

Exceptions

Table 8–7 datatype_ARG Procedure Exceptions

Exception	Description
ORA-23323	Argument value is too long.

TRANSACTION procedure

This procedure indicates the start of a new deferred transaction. If you omit this call, then Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

Syntax

```
DBMS_DEFER.TRANSACTION (
   nodes  IN   node_list_t);
```

Parameters

Table 8–8 TRANSACTION Procedure Parameters

Parameter	Description
nodes	A PL/SQL 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 until a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive.

Exceptions

Table 8–9 TRANSACTION Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous transaction was not correctly formed or terminated.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Raised by DBMS_DEFER.CALL if the node list contains duplicates.

Usage Notes

The TRANSACTION 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.

DBMS_DEFER_QUERY Package

Summary of Subprograms

Table 8–10 DBMS_DEFER_QUERY Package Subprograms

Subprogram	Description
GET_ARG_FORM function on page 8-12	Determines the form of an argument in a deferred call.
GET_ARG_TYPE function on page 8-13	Determines the type of an argument in a deferred call.
GET_CALL_ARGS procedure on page 8-15	Returns the text version of the various arguments for the given call.
GET_datatype_ARG function on page 8-16	Determines the value of an argument in a deferred call.

GET_ARG_FORM function

This function determines the form of an argument in a deferred call. This function will return the character set ID of a deferred call parameter.

For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

Syntax

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

Parameters

Table 8–11 GET_ARG_FORM Function Parameters

Parameter	Description
callno	Call identifier from the DEFCALL view.
arg_no	Position of desired parameter in calls argument list. Parameter positions are 1..*number* of parameters in call.
deferred_tran_id	Deferred transaction ID.

Exceptions

Table 8–12 GET_ARG_FORM Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 8–13 GET_ARG_Form Function Returns

Return Value	Corresponding Datatype
1	CHAR, VARCHAR2, CLOB
2	NCHAR, NVARCHAR2, NCLOB

GET_ARG_TYPE function

This function determines the type of an argument in a deferred call. The type of the deferred RPC parameter will be returned.

For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

Syntax

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

Parameters

Table 8–14 GET_ARG_TYPE Function Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose type you want to determine. The first argument to a procedure is in position 1.
deferred_tran_id	Identifier of the deferred transaction.

Exceptions

Table 8–15 GET_ARG_TYPE Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 8–16 GET_ARG_TYPE Function Returns

Return Value	Corresponding Datatype
1	VARCHAR2
2	NUMBER
11	ROWID
12	DATE
23	RAW
96	CHAR
112	CLOB
113	BLOB

GET_CALL_ARGS procedure

This procedure returns the text version of the various arguments for the given call. The text version is limited to the first 2000 bytes.

Syntax

```
DBMS_DEFER_QUERY.GET_CALL_ARGS (
   callno    IN  NUMBER,
   startarg  IN  NUMBER := 1,
   argcnt    IN  NUMBER,
   argsize   IN  NUMBER,
   tran_id   IN  VARCHAR2,
   date_fmt  IN  VARCHAR2,
   types     OUT TYPE_ARY,
   forms     OUT TYPE_ARY,
   vals      OUT VAL_ARY);
```

Parameters

Table 8–17 GET_CALL_ARGS Procedure Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred RPC.
startarg	Numerical position of the first argument you want described.
argcnt	Number of arguments in the call.
argsize	Maximum size of returned argument.
tran_id	Identifier of the deferred transaction.
date_fmt	Format in which the date should be returned.
types	Array containing the types of arguments.
forms	Array containing the character set forms of arguments.
vals	Array containing the values of the arguments in a textual form.

Exceptions

Table 8–18 GET_CALL_ARGS Procedure Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

GET_*datatype*_ARG function

This function determines the value of an argument in a deferred call.

For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

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*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| ROWID
| BLOB
| CLOB
| NCLOB
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 8–19 GET_datatype_ARG Function Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose value you want to determine. The first argument to a procedure is in position one.
deferred_tran_id	Identifier of the deferred transaction. Defaults to the last transaction identifier passed to GET_ARG_TYPE. The default is NULL.

Exceptions

Table 8–20 GET_datatype_ARG Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.
ORA-26564	Argument in this position is not of the specified type.

DBMS_DEFER_SYS Package

Summary of Subprograms

Table 8–21 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
ADD_DEFAULT_DEST procedure on page 8-20	Adds a destination database to the DEFDEFAULTDEST view.
DELETE_DEFAULT_DEST procedure on page 8-21	Removes a destination database from the DEFDEFAULTDEST view.
DELETE_DEF_DESTINATION procedure on page 8-22	Removes a destination database from the DEFSCHEDULE view.
DELETE_ERROR on page 8-23	Deletes a transaction from the DEFERROR view.
DELETE_TRAN on page 8-24	Deletes a transaction from the DEFTRANDEST view.
DISABLED on page 8-25	Determines whether propagation of the deferred transaction queue from the current site to a given site is enabled.
EXCLUDE_PUSH on page 8-26	Acquires an exclusive lock that prevents deferred transaction PUSH.
EXECUTE_ERROR on page 8-27	Re-executes a deferred transaction that did not initially complete successfully.
EXECUTE_ERROR_AS_USER on page 8-28	Re-executes a deferred transaction that did not initially complete successfully.
PURGE on page 8-29	Purges pushed transactions from the deferred transaction queue at your current master or snapshot site.
PUSH function on page 8-31	Forces a deferred remote procedure call queue at your current master or snapshot site to be pushed to another master site.
REGISTER_PROPAGATOR procedure on page 8-34	Registers the given user as the propagator for the local database.

Table 8–21 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
SCHEDULE_PURGE procedure on page 8-35	Schedules a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site.
SCHEDULE_PUSH procedure on page 8-37	Schedules a job to push the deferred transaction queue to a remote master destination.
SET_DISABLED procedure on page 8-39	Disables or enables propagation of the deferred transaction queue from the current site to a given destination site.
UNREGISTER_PROPAGATOR procedure on page 8-40	Unregister a user as the propagator from the local database.
UNSCHEDULE_PURGE procedure on page 8-41	Stops automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.
UNSCHEDULE_PUSH procedure on page 8-42	Stops automatic pushes of the deferred transaction queue from a snapshot or master site to another master site.

ADD_DEFAULT_DEST procedure

This procedure adds a destination database to the DEFDEFAULTDEST view.

Syntax

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST (
   dblink   IN   VARCHAR2);
```

Parameters

Table 8–22 ADD_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to add to the DEFDEFAULTDEST view.

Exceptions

Table 8–23 ADD_DEFAULT_DEST Procedure Exceptions

Exception	Description
ORA-23352	The dblink that you specified is already in the default list.

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 8–24 DELETE_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to delete from the DEFDEFAULTDEST view. If Oracle does not find this dblink in the view, then no action is taken.

DELETE_DEF_DESTINATION procedure

This procedure removes a destination database from the DEFSCHEDULE view.

Syntax

```
DBMS_DEFER_SYS.DELETE_DEF_DESTINATION (
   destination   IN   VARCHAR2,
   force         IN   BOOLEAN := FALSE);
```

Parameters

Table 8–25 DELETE_DEF_DESTINATION Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the destination that you want to delete from the DefSchedule view. If Oracle does not find this destination in the view, then no action is taken.
force	When set to TRUE, Oracle ignores all safety checks and deletes the destination.

DELETE_ERROR

To delete a transaction from the DEFERROR view.

Syntax

```
DBMS_DEFER_SYS.DELETE_ERROR(
   deferred_tran_id     IN   VARCHAR2,
   destination          IN   VARCHAR2);
```

Parameters

Table 8–26 Parameters for DELETE_ERROR

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to remove from the DEFERROR view. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed from the DEFERROR view.

DELETE_TRAN

To delete 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 8–27 Parameters for DELETE_TRAN

Parameter	Description
deferred_tran_id	ID number from the DEFTRAN view of the deferred transaction that you want to delete. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.
destination	The fully qualified database name from the DEFTRANDEST view of the database to which the transaction was originally queued. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.

DISABLED

To determine whether propagation of the deferred transaction queue from the current site to a given site is enabled. The DISABLED function returns TRUE if the deferred remote procedure call (RPC) queue is disabled for the given destination.

Syntax

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

Parameters

Table 8–28 Parameter for DISABLED

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to check.

Returns

Table 8–29 Return Values for DISABLED

Value	Description
TRUE	Propagation to this site from the current site is disabled.
FALSE	Propagation to this site from the current site is enabled.

Exceptions

Table 8–30 Exception for DISABLED

Exception	Description
NO_DATA_FOUND	DESTINATION does not appear in the DEFSCHEDULE view.

EXCLUDE_PUSH

To acquire an exclusive lock that prevents deferred transaction PUSH (either serial or parallel). This function does a commit when acquiring the lock. The lock is acquired with RELEASE_ON_COMMIT => TRUE, so that pushing of the deferred transaction queue can resume after the next commit.

Syntax

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

Parameters

Table 8–31 Parameter for EXCLUDE_PUSH

Parameter	Description
timeout	Timeout in seconds. If the lock cannot be acquired within this time period (either because of an error or because a PUSH is currently under way), then the call returns a value of 1. A timeout value of DBMS_LOCK.MAXWAIT waits indefinitely.

Returns

E_PUSH

Table 8–32 Return Values for EXCLUDE_PUSH

Value	Description
0	Success, lock acquired.
1	Timeout, no lock acquired.
2	Deadlock, no lock acquired.
4	Already own lock.

EXECUTE_ERROR

To reexecute a deferred transaction that did not initially complete successfully. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

Syntax

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

Parameters

Table 8–33 Parameters for EXECUTE_ERROR

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this is NULL, then all transactions queued for destination are re-executed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.

Exceptions

Table 8–34 Exceptions for EXECUTE_ERROR

Exception	Description
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

EXECUTE_ERROR_AS_USER

To reexecute a deferred transaction that did not initially complete successfully. Each transaction is executed in the security context of the connected user. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

Syntax

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

Parameters

Table 8–35 Parameters for EXECUTE_ERROR_AS_USER

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this is NULL, then all transactions queued for destination are re-executed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.

Exceptions

Table 8–36 Exceptions for EXECUTE_ERROR_AS_USER

Exception	Description
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

PURGE

To purge pushed transactions from the deferred transaction queue at your current master or snapshot site.

Syntax

```
DBMS_DEFER_SYS.PURGE (
   purge_method         IN  BINARY_INTEGER := purge_method_quick,
   rollback_segment     IN  VARCHAR2       := NULL,
   startup_seconds      IN  BINARY_INTEGER := 0,
   execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
   delay_seconds        IN  BINARY_INTEGER := 0,
   transaction_count    IN  BINARY_INTEGER := transactions_infinity,
   write_trace          IN  BOOLEAN        := NULL);
  RETURN BINARY_INTEGER;
```

Parameters

Table 8–37 Parameters for PURGE

Parameter	Description
purge_method	Controls how to purge the deferred transaction queue; `purge_method_quick` costs less, while `purge_method_precise` offers better precision.
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 shutdown 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.

Returns

Table 8–38 Return Values for Purge

Value	Description
0	OK, terminated after delay_seconds expired.
1	Terminated by lock timeout while starting.
2	Terminated by exceeding execution_seconds.
3	Terminated by exceeding transaction_count.
5	Terminated after errors.

Exceptions

Table 8–39 Exceptions for PURGE

Exception	Description
argoutofrange	Parameter value is out of a valid range.
executiondisabled	Execution of purging is disabled.
defererror	Internal error.

PUSH function

This function forces a deferred remote procedure call queue at your current master or snapshot site to be pushed (executed, propagated) to another master site using either serial or parallel propagation.

Syntax

```
DBMS_DEFER_SYS.PUSH (
   destination          IN  VARCHAR2,
   parallelism          IN  BINARY_INTEGER := 0,
   heap_size            IN  BINARY_INTEGER := 0)
   stop_on_error        IN  BOOLEAN        := FALSE,
   write_trace          IN  BOOLEAN        := FALSE,
   startup_seconds      IN  BINARY_INTEGER := 0,
   execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
   delay_seconds        IN  BINARY_INTEGER := 0,
   transaction_count    IN  BINARY_INTEGER := transactions_infinity,
   delivery_order_limit IN  NUMBER         := delivery_order_infinity)
  RETURN BINARY_INTEGER;
```

Parameters

Table 8–40 PUSH Function Parameters

Parameter	Description
destination	The fully qualified database name of the master to which you are forwarding changes.
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.
stop_on_error	The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, then shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site.
write_trace	When set to TRUE, Oracle records the result value returned by the function in the server's trace file.

Table 8–40 PUSH Function Parameters

Parameter	Description
startup_seconds	Maximum number of seconds to wait for a previous push to the same destination.
execution_seconds	If >0, then stop push cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.
delivery_order_limit	Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit

Returns

PUSH

Table 8–41 PUSH Function Returns

Value	Description
0	OK, terminated after delay_seconds expired.
1	Terminated by lock timeout while starting.
2	Terminated by exceeding execution_seconds.
3	Terminated by exceeding transaction_count.
4	Terminated by exceeding delivery_order_limit.
5	Terminated after errors.

Exceptions

Table 8–42 Exceptions for PUSH

Exception	Description
deferror incompleteparallelpush	Serial propagation requires that parallel propagation shuts down cleanly.
executiondisabled	Execution of deferred RPCs is disabled at the destination.
crt_err_err	Error while creating entry in DEFERROR.
deferred_rpc_qiesce	Replication activity for object group is suspended.
commfailure	Communication failure during deferred RPC.
missingpropator	A propagator does not exist.

REGISTER_PROPAGATOR procedure

This procedure registers the given user as the propagator for the local database. It also grants to the given user CREATE SESSION, CREATE PROCEDURE, CREATE DATABASE LINK, and EXECUTE ANY PROCEDURE privileges (so that the user can create wrappers).

Syntax

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

Parameters

Table 8–43 REGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the user.

Exceptions

Table 8–44 REGISTER_PROPAGATOR Procedure Exceptions

Exception	Description
missinguser	Given user does not exist.
alreadypropagator	Given user is already the propagator.
duplicatepropagator	There is already a different propagator.

SCHEDULE_PURGE procedure

This procedure schedules a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site. You should schedule one purge job.

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PURGE (
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := NULL,
   purge_method         IN  BINARY_INTEGER := NULL,
   rollback_segment     IN  VARCHAR2       := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL,
   write_trace          IN  BOOLEAN        := NULL);
```

Parameters

Table 8–45 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
interval	Allows you to provide a function to calculate the next time to purge. This value is stored in the interval field of the DEFSCHEDULE view and calculates the next_date field of this view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If the field had no previous value, it is created with a value of NULL. If you do not supply a value for this field, you must supply a value for next_date.
next_date	Allows you to specify a given time to purge pushed transactions from the site's queue. This value is stored in the next_date field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If this field had no previous value, it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for interval.
reset	Set to TRUE to reset LAST_TXN_COUNT, LAST_ERROR, and LAST_MSG to NULL.

Table 8–45 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
purge_method	Controls how to purge the deferred transaction queue; purge_method_quick costs less, while purge_method_precise offers better precision.
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 shutdown cleanly after purging transaction_count number of transactions.
write_trace	When set to TRUE, Oracle records the result value returned by the PURGE function in the server's trace file.

SCHEDULE_PUSH procedure

This procedure schedules a job to push the deferred transaction queue to a remote master destination. This procedure does a COMMIT.

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PUSH (
   destination          IN  VARCHAR2,
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := FALSE,
   parallelism          IN  BINARY_INTEGER := NULL,
   heap_size            IN  BINARY_INTEGER := NULL,
   stop_on_error        IN  BOOLEAN        := NULL,
   write_trace          IN  BOOLEAN        := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL);
```

Parameters

Table 8–46 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the master to which you are forwarding changes.
interval	Allows you to provide a function to calculate the next time to push. This value is stored in the interval field of the DEFSCHEDULE view and calculates the next_date field of this view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If the field had no previous value, it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for next_date.
next_date	Allows you to specify a given time to push deferred transactions to the master site destination. This value is stored in the next_date field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If this field had no previous value, then it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for interval.

Table 8–46 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
reset	Set to TRUE to reset LAST_TXN_COUNT, LST_ERROR, and LAST_MSG to NULL.
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.
stop_on_error	The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, then shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site.
write_trace	When set to TRUE, Oracle records the result value returned by the function in the server's trace file.
startup_seconds	Maximum number of seconds to wait for a previous push to the same destination.
execution_seconds	If >0, then stop execution cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.

SET_DISABLED procedure

To disable or enable propagation of the deferred transaction queue from the current site to a given destination site. If the disabled parameter is TRUE, then the procedure disables propagation to the given 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 given 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 given destination and, although this does not push the queue, it permits future invocations to PUSH to push the queue to the given 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);
```

Parameters

Table 8–47 SET_DISABLED Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to change.
disabled	By default, this parameter disables propagation of the deferred transaction queue from your current site to the given destination. Set this to FALSE to enable propagation.

Exceptions

Table 8–48 SET_DISABLED Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the given destination.

UNREGISTER_PROPAGATOR procedure

To unregister a user as the propagator from the local database. This procedure

- Deletes the given propagator from DEFPROPAGATOR.
- Revokes privileges granted by REGISTER_PROPAGATOR from the given user (including identical privileges granted independently).
- Drops any generated wrappers in the schema of the given 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 8–49 UNREGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the propagator user.
timeout	Timeout in seconds. If the propagator is in use, then the procedure waits until timeout. The default is DBMS_LOCK.MAXWAIT.

Exceptions

Table 8–50 UNREGISTER_PROPAGATOR Procedure Exceptions

Parameter	Description
missingpropagator	Given user is not a propagator.
propagator_inuse	Propagator is in use, and thus cannot be unregistered. Try later.

UNSCHEDULE_PURGE procedure

This procedure stops automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PURGE;
```

Parameters

None

UNSCHEDULE_PUSH procedure

This procedure stops automatic pushes of the deferred transaction queue from a snapshot or master site to another master site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PUSH (
   dblink   IN   VARCHAR2);
```

Parameters

Table 8–51 UNSCHEDULE_PUSH Procedure Parameters

Parameter	Description
dblink	Fully qualified pathname to master database site at which you want to unschedule periodic execution of deferred remote procedure calls.

Table 8–52 UNSCHEDULE_PUSH Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the given dblink.

DBMS_OFFLINE_OG Package

Summary of Subprograms

Table 8–53 DBMS_OFFLINE_OG Package Subprograms

Subprogram	Description
BEGIN_INSTANTIATION procedure on page 8-44	Starts offline instantiation of a replicated master group.
BEGIN_LOAD procedure on page 8-46	Disables triggers while data is imported to new master site as part of offline instantiation.
END_INSTANTIATION procedure on page 8-48	Completes offline instantiation of a replicated master group.
END_LOAD procedure on page 8-50	Re-enables triggers after importing data to new master site as part of offline instantiation.
RESUME_SUBSET_OF_MASTERS procedure on page 8-52	Resumes replication activity at all existing sites except the new site during offline instantiation of a replicated master group.

BEGIN_INSTANTIATION procedure

This procedure starts offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Note: This procedure is used in performing 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 snapshot) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 8–54 BEGIN_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to replicate to the new site.
new_site	The fully qualified database name of the new site to which you want to replicate the object group.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–55 BEGIN_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
dbms_repcat.nonmasterdef	This procedure must be called from the master definition site.
sitealreadyexists	Given site is already a master site for this object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.missing_flavor	If you receive this exception, contact Oracle Worldwide Support.

BEGIN_LOAD procedure

This procedure disables triggers while data is imported to new master site as part of offline instantiation. You must call this procedure from the new master site.

Note: This procedure is used in performing 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 snapshot) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 8–56 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group whose members you are importing.
new_site	The fully qualified database name of the new site at which you will be importing the object group members.

Exceptions

Table 8–57 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of the new master site must be QUIESCED.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.

END_INSTANTIATION procedure

This procedure completes offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Note: This procedure is used in performing 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 snapshot) 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 8–58 END_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the object group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the object group.

Exceptions

Table 8–59 END_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group or new master site name.
dbms_repcat. nonmasterdef	This procedure must be called from the master definition site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.

END_LOAD procedure

This procedure re-enables triggers after importing data to new master site as part of offline instantiation. You must call this procedure from the new master site.

Note: This procedure is used in performing 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 snapshot) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.END_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 8–60 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group whose members you have finished importing.
new_site	The fully qualified database name of the new site at which you have imported the object group members.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–61 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of the new master site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.flavor_noobject	If you receive this exception, contact Oracle Worldwide Support.
dbms_repcat.flavor_contains	If you receive this exception, contact Oracle Worldwide Support.

RESUME_SUBSET_OF_MASTERS procedure

This procedure resumes replication activity at all existing sites except the new site during offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Note: This procedure is used in performing 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 snapshot) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2
   override  IN  BOOLEAN := FALSE);
```

Parameters

Table 8–62 RESUME_SUBSET_OF_MASTERS Procedure Parameters

Parameter	Description
gname	Name of the object group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the object group.
override	If this is TRUE, then it ignores any pending RepCat administration 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 administration request for gname at that master.

Exceptions

Table 8–63 RESUME_SUBSET_OF_MASTERS Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
dbms_repcat. nonmasterdef	This procedure must be called from the master definition site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.

DBMS_OFFLINE_SNAPSHOT Package

Summary of Subprograms

Table 8–64 DBMS_OFFLINE_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_LOAD procedure on page 8-55	Prepares a snapshot site for import of a new snapshot as part of offline instantiation.
END_LOAD procedure on page 8-57	Completes offline instantiation of a snapshot.

BEGIN_LOAD procedure

This procedure prepares a snapshot site for import of a new snapshot as part of offline instantiation. You must call this procedure from the snapshot site for the new snapshot.

Note: This procedure is used in performing an offline instantiation of a snapshot.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
   gname               IN   VARCHAR2,
   sname               IN   VARCHAR2,
   master_site         IN   VARCHAR2,
   snapshot_oname      IN   VARCHAR2,
   storage_c           IN   VARCHAR2 := '',
   comment             IN   VARCHAR2 := '',
   min_communication   IN   BOOLEAN  := TRUE);
```

Parameters

Table 8–65 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group for the snapshot that you are creating using offline instantiation.
sname	Name of the schema for the new snapshot.
master_site	Fully qualified database name of the snapshot's master site.
snapshot_oname	Name of the temporary snapshot created at the master site.
storage_c	Storage options to use when creating the new snapshot at the snapshot site.
comment	User comment.
min_communication	If TRUE, then the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group.

Exceptions

Table 8–66 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group, schema, master site, or snapshot name.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.
missingremotesnap	Could not locate given snapshot at given master site.
dbms_repcat. missingschema	Given schema does not exist.
snaptabmismatch	Base table name of the snapshot at the master and snapshot do not match.

END_LOAD procedure

This procedure completes offline instantiation of a snapshot. You must call this procedure from the snapshot site for the new snapshot.

Note: This procedure is used in performing an offline instantiation of a snapshot.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
   gname           IN  VARCHAR2,
   sname           IN  VARCHAR2,
   snapshot_oname  IN  VARCHAR2);
```

Parameters

Table 8–67 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group for the snapshot that you are creating using offline instantiation.
sname	Name of the schema for the new snapshot.
snapshot_oname	Name of the snapshot.

Exceptions

Table 8–68 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group, schema, or snapshot name.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.nonsnapshot	This procedure must be called from the snapshot site.

DBMS_RECTIFIER_DIFF Package

Summary of Subprograms

Table 8–69 DBMS_RECTIFIER_DIFF Package Subprograms

Subprogram	Description
DIFFERENCES procedure on page 8-60	Determines the differences between two tables.
RECTIFY procedure on page 8-63	Resolves the differences between two tables.

DIFFERENCES procedure

This procedure determines the differences between two tables.

Syntax

```
DBMS_RECTIFIER_DIFF.DIFFERENCES (
   sname1               IN  VARCHAR2,
   oname1               IN  VARCHAR2,
   reference_site       IN  VARCHAR2 := '',
   sname2               IN  VARCHAR2,
   oname2               IN  VARCHAR2,
   comparison_site      IN  VARCHAR2 := '',
   where_clause         IN  VARCHAR2 := '',
   { column_list        IN  VARCHAR2 := '',
   | array_columns      IN  dbms_utility.name_array, }
   missing_rows_sname   IN  VARCHAR2,
   missing_rows_oname1  IN  VARCHAR2,
   missing_rows_oname2  IN  VARCHAR2,
   missing_rows_site    IN  VARCHAR2 := '',
   max_missing          IN  INTEGER,
   commit_rows          IN  INTEGER := 500);
```

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

Parameters

Table 8–70 DIFFERENCES Procedure Parameters

Parameter	Description
sname1	Name of the schema at REFERENCE_SITE.
oname1	Name of the table at REFERENCE_SITE.
reference_site	Name of the reference database site. The default, NULL, indicates the current site.
sname2	Name of the schema at COMPARISON_SITE.
oname2	Name of the table at COMPARISON_SITE.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.

Table 8–70 DIFFERENCES Procedure Parameters

Parameter	Description
where_clause	Only rows satisfying this restriction are selected for comparison. The default, NULL, indicates the current site.
column_list	A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared.
array_columns	A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, then all columns are used.
missing_rows_sname	Name of the schema containing the tables with the missing rows.
missing_rows_oname1	Name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent.
missing_rows_site	Name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site.
max_missing	Integer that refers to the maximum number of rows that should be inserted into the missing_rows_oname table. If more than max_missing number of rows is missing, then that many rows are inserted into missing_rows_oname, and the routine then returns normally without determining whether more rows are missing; this argument is useful in the cases that the fragments are so different that the missing rows table will have too many entries and there's no point in continuing. Raises exception badnumber if max_missing is less than 1 or NULL.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted.

Exceptions

Table 8–71 DIFFERENCES Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	COMMIT_ROWS parameter less than 1.
missingprimarykey	Column list must include primary key (or SET_COLUMNS equivalent).
badname	NULL or empty string for table or schema name.
cannotbenull	Parameter cannot be NULL.
notshapeequivalent	Tables being compared are not shape equivalent. Shape refers to the number of columns, their column names, and the column datatypes.
unknowncolumn	Column does not exist.
unsupportedtype	Type not supported.
dbms_repcat. commfailure	Remote site is inaccessible.
dbms_repcat. missingobject	Table does not exist.

Restrictions

The error ORA-00001 (Unique constraint violated) is issued when there are any unique or primary key constraints on the MISSING_ROWS_DATA table.

RECTIFY procedure

This procedure resolves the differences between two tables.

Syntax

```
DBMS_RECTIFIER_DIFF.RECTIFY (
   sname1               IN  VARCHAR2,
   oname1               IN  VARCHAR2,
   reference_site       IN  VARCHAR2 := '',
   sname2               IN  VARCHAR2,
   oname2               IN  VARCHAR2,
   comparison_site      IN  VARCHAR2 := '',
   { column_list        IN  VARCHAR2 := '',
   | array_columns      IN  dbms_utility.name_array, }
   missing_rows_sname   IN  VARCHAR2,
   missing_rows_oname1  IN  VARCHAR2,
   missing_rows_oname2  IN  VARCHAR2,
   missing_rows_site    IN  VARCHAR2 := '',
   commit_rows          IN  INTEGER := 500);
```

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

Parameters

Table 8–72 RECTIFY Procedure Parameters

Parameter	Description
sname1	Name of the schema at REFERENCE_SITE.
oname1	Name of the table at REFERENCE_SITE.
reference_site	Name of the reference database site. The default, NULL, indicates the current site.
sname2	Name of the schema at COMPARISON_SITE.
oname2	Name of the table at COMPARISON_SITE.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.
column_list	A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared.

Table 8–72 RECTIFY Procedure Parameters

Parameter	Description
array_columns	A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, then all columns are used.
missing_rows_sname	Name of the schema containing the tables with the missing rows.
missing_rows_oname1	Name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent.
missing_rows_site	Name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted.

Exceptions

Table 8–73 RECTIFY Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	COMMIT_ROWS parameter less than 1.
badname	NULL or empty string for table or schema name.
dbms_repcat.commfailure	Remote site is inaccessible.
dbms_repcat.missingobject	Table does not exist.

DBMS_REFRESH Package

Summary of Subprograms

Table 8–74 DBMS_REFRESH Package Subprograms

Subprogram	Description
ADD procedure on page 8-66	Adds snapshots to a refresh group.
CHANGE procedure on page 8-67	Changes the refresh interval for a snapshot group.
DESTROY procedure on page 8-69	Removes all of the snapshots from a refresh group and deletes the refresh group.
MAKE procedure on page 8-70	Specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.
REFRESH procedure on page 8-73	Manually refreshes a refresh group.
SUBTRACT procedure on page 8-74	Removes snapshots from a refresh group.

ADD procedure

This procedure adds snapshots to a refresh group.

For additional information, see "ADD OBJECTS TO REFRESH GROUP" on page 5-6. Also see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.ADD (
   name      IN VARCHAR2,
   { list    IN VARCHAR2,
   | tab     IN DBMS_UTILITY.UNCL_ARRAY, }
   lax       IN BOOLEAN := FALSE);
```

> **Note:** This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 8–75 ADD Procedures Parameters

Parameter	Description
name	Name of the refresh group to which you want to add members.
list	Comma-separated list of snapshots that you want to add to the refresh group. (Synonyms are not supported.)
tab	Instead of a comma-separated list, you can supply a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a snapshot. The first snapshot should be in position 1. The last position must be NULL.
lax	A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from one group to another, then you must set the lax flag to TRUE to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to ADD generates an error message.

CHANGE procedure

This procedure changes the refresh interval for a snapshot group.

For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.CHANGE (
   name                 IN VARCHAR2,
   next_date            IN DATE           := NULL,
   interval             IN VARCHAR2       := NULL,
   implicit_destroy     IN BOOLEAN        := NULL,
   rollback_seg         IN VARCHAR2       := NULL,
   push_deferred_rpc    IN BOOLEAN        := NULL,
   refresh_after_errors IN BOOLEAN        := NULL,
   purge_option         IN BINARY_INTEGER := NULL,
   parallelism          IN BINARY_INTEGER := NULL,
   heap_size            IN BINARY_INTEGER := NULL);
```

Parameters

Table 8–76 CHANGE Procedures Parameters

Parameter	Description
name	Name of the refresh group for which you want to alter the refresh interval.
next_date	Next date that you want a refresh to occur. By default, this date remains unchanged.
interval	Function used to calculate the next time to refresh the snapshots in the group. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh. By default, the interval remains unchanged.
implicit_destroy	Allows you to reset the value of the `implicit_destroy` flag. If this flag is set, then Oracle automatically deletes the group if it no longer contains any members. By default, this flag remains unchanged.

Table 8–76 CHANGE Procedures Parameters

Parameter	Description
rollback_seg	Allows you to change the rollback segment used. By default, the rollback segment remains unchanged. To reset this parameter to use the default rollback segment, specify NULL, including the quotes. Specifying NULL without quotes indicates that you do not want to change the rollback segment currently being used.
push_deferred_rpc	Used by updatable snapshots only. Set this parameter to TRUE if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost. By default, this flag remains unchanged
refresh_after_ errors	Used by updatable snapshots 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 snapshot's master. By default, this flag remains unchanged.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 = don't 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 *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 = serial propagation; $n > 0$ = parallel propagation with n parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.

DESTROY procedure

This procedure removes all of the snapshots from a refresh group and delete the refresh group.

For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.DESTROY (
   name   IN   VARCHAR2);
```

Parameters

Table 8–77 DESTROY Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to destroy.

MAKE procedure

This procedure specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.

For additional information, see "CREATE REFRESH GROUP" on page 5-4. Also see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

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.

Table 8–78 MAKE Procedure Parameters

Parameter	Description
name	Unique name used to identify the refresh group. Refresh groups must follow the same naming conventions as tables.
list	Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database.

Table 8–78 MAKE Procedure Parameters

Parameter	Description
tab	Instead of a comma separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of N snapshots, then the first snapshot should be in position 1 and the N + 1 position should be set to NULL.
next_date	Next date that you want a refresh to occur.
interval	Function used to calculate the next time to refresh the snapshots in the group. This field is used with the NEXT_DATE value. For example, if you specify NEXT_DAY(SYSDATE+1, "MONDAY") as your interval, and if your NEXT_DATE evaluates to Monday, then Oracle refreshes the snapshots every Monday. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh.
implicit_destroy	Set this to TRUE if you want to delete the refresh group automatically when it no longer contains any members. Oracle checks this flag only when you call the SUBTRACT procedure. That is, setting this flag still allows you to create an empty refresh group.
lax	A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from an existing group to a new refresh group, then you must set this to TRUE to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to MAKE generates an error message.
job	Needed by the Import utility. Use the default value, 0.
rollback_seg	Name of the rollback segment to use while refreshing snapshots. The default, NULL, uses the default rollback segment.
push_deferred_rpc	Used by updatable snapshots only. Use the default value, TRUE, if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	Used by updatable snapshots 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 snapshot's master.

Table 8–78 MAKE Procedure Parameters

Parameter	Description
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 = don't 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 *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set this unless so directed by Oracle Worldwide Support.

REFRESH procedure

This procedure manually refreshes a refresh group.

For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.REFRESH (
   name    IN     VARCHAR2);
```

Table 8–79 REFRESH Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to refresh manually.

SUBTRACT procedure

This procedure removes snapshots from a refresh group.

For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.SUBTRACT (
   name       IN    VARCHAR2,
   { list     IN    VARCHAR2,
   | tab      IN     DBMS_UTILITY.UNCL_ARRAY, }
   lax        IN     BOOLEAN := FALSE);
```

> **Note:** This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 8–80 SUBTRACT Procedure Parameters

Parameter	Description
name	Name of the refresh group from which you want to remove members.
list	Comma-separated list of snapshots that you want to remove from the refresh group. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database.
tab	Instead of a comma-separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of N snapshots, then the first snapshot should be in position 1 and the N+1 position should be set to NULL.
lax	Set this to FALSE if you want Oracle to generate an error message if the snapshot you are attempting to remove is not a member of the refresh group.

DBMS_REPCAT Package

Summary of Subprograms

Table 8–81 DBMS_REPCAT Package Subprograms

Subprogram	Description
ADD_GROUPED_COLUMN procedure on page 8-79	Adds members to an existing column group.
ADD_MASTER_DATABASE procedure on page 8-80	Adds another master site to your replicated environment.
ADD_PRIORITY_datatype procedure on page 8-82	Adds a member to a priority group.
ADD_SITE_PRIORITY_SITE procedure on page 8-84	Adds a new site to a site priority group.
ADD_conflicttype_RESOLUTION procedure on page 8-85	Designates a method for resolving an update, delete, or uniqueness conflict.
ALTER_MASTER_PROPAGATION procedure on page 8-89	Alters the propagation method for a given object group at a given master site.
ALTER_MASTER_REPOBJECT procedure on page 8-91	Alters an object in your replicated environment.
ALTER_PRIORITY procedure on page 8-93	Alters the priority level associated with a given priority group member.
ALTER_PRIORITY_datatype procedure on page 8-94	Alters the value of a member in a priority group.
ALTER_SITE_PRIORITY procedure on page 8-96	Alters the priority level associated with a given site.
ALTER_SITE_PRIORITY_SITE procedure on page 8-97	Alters the site associated with a given priority level.
ALTER_SNAPSHOT_PROPAGATION procedure on page 8-98	Alters the propagation method for a given object group at the current snapshot site.
CANCEL_STATISTICS procedure on page 8-99	Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.
COMMENT_ON_COLUMN_GROUP procedure on page 8-100	Updates the comment field in the RepColumn_Group view for a column group.

Table 8–81 DBMS_REPCAT Package Subprograms

Subprogram	Description
COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY procedure on page 8-101	Updates the comment field in the REPPRIORITY_GROUP view for a (site) priority group.
COMMENT_ON_REPGROUP procedure on page 8-102	Updates the comment field in the REPGROUP view for a replicated master group.
COMMENT_ON_REPSITES procedure on page 8-103	Updates the comment field in the RepSite view for a replicated site.
COMMENT_ON_REPOBJECT procedure on page 8-104	Updates the comment field in the RepObject view for a replicated object.
COMMENT_ON_conflicttype_RESOLUTION procedure on page 8-105	Updates the comment field in the RepResolution view for a conflict resolution routine.
CREATE_MASTER_REPGROUP procedure on page 8-109	Creates a new, empty, quiesced master replication object group.
CREATE_MASTER_REPOBJECT procedure on page 8-110	Indicates that an object is a replicated object.
CREATE_SNAPSHOT_REPGROUP procedure on page 8-114	Creates a new, empty snapshot replication object group in your local database.
CREATE_SNAPSHOT_REPOBJECT procedure on page 8-116	Adds a replicated object to your snapshot site.
DEFINE_COLUMN_GROUP procedure on page 8-118	Creates an empty column group
DEFINE_PRIORITY_GROUP procedure on page 8-119	Creates a new priority group for a replicated master group.
DEFINE_SITE_PRIORITY procedure on page 8-121	Creates a new site priority group for a replicated master group.
DO_DEFERRED_REPCAT_ADMIN procedure on page 8-122	Executes the local outstanding deferred administrative procedures for the given replicated master group at the current master site, or for all master sites.
DROP_COLUMN_GROUP procedure on page 8-123	Drops a column group.
DROP_GROUPED_COLUMN procedure on page 8-124	Removes members from a column group.

Table 8–81 DBMS_REPCAT Package Subprograms

Subprogram	Description
DROP_MASTER_REPGROUP procedure on page 8-125	Drops a replicated master group from your current site.
DROP_MASTER_REPOBJECT procedure on page 8-127	Drops a replicated object from a replicated master group.
DROP_PRIORITY procedure on page 8-128	Drops a member of a priority group by priority level.
DROP_PRIORITY_GROUP procedure on page 8-129	Drops a priority group for a given replicated master group.
DROP_PRIORITY_datatype procedure on page 8-130	Drops a member of a priority group by value.
DROP_SITE_PRIORITY procedure on page 8-132	Drops a site priority group for a given replicated master group.
DROP_SITE_PRIORITY_SITE procedure on page 8-133	Drops a given site, by name, from a site priority group.
DROP_SNAPSHOT_REPGROUP procedure on page 8-134	Drops a snapshot site from your replicated environment.
DROP_SNAPSHOT_REPOBJECT procedure on page 8-135	Drops a replicated object from a snapshot site.
DROP_conflicttype_ RESOLUTION procedure on page 8-136	Drops an update, delete, or uniqueness conflict resolution routine.
EXECUTE_DDL procedure on page 8-138	Supplies DDL that you want to have executed at each master site.
GENERATE_REPLICATION_ SUPPORT procedure on page 8-140	Generates the triggers, packages, and procedures needed to support replication.
GENERATE_SNAPSHOT_SUPPORT procedure on page 8-142	Activates triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.
MAKE_COLUMN_GROUP procedure on page 8-144	Creates a new column group with one or more members.
PURGE_MASTER_LOG procedure on page 8-146	Removes local messages in the RepCatLog associated with a given identification number, source, or replicated master group.

Table 8–81 DBMS_REPCAT Package Subprograms

Subprogram	Description
PURGE_STATISTICS procedure on page 147	Removes information from the RepResolution_Statistics view.
REFRESH_SNAPSHOT_REPGROUP procedure on page 8-148	Refreshes a snapshot site object group with the most recent data from its associated master site.
REGISTER_SNAPSHOT_REPGROUP procedure on page 8-149	Facilitates the administration of snapshots at their respective master sites by inserting/modifying/deleting from repcat_repsite.
REGISTER_STATISTICS procedure on page 8-150	Collects information about the successful resolution of update, delete and uniqueness conflicts for a table.
RELOCATE_MASTERDEF procedure on page 151	Changes your master definition site to another master site in your replicated environment.
REMOVE_MASTER_DATABASES procedure on page 8-153	Removes one or more master databases from a replicated environment.
REPCAT_IMPORT_CHECK procedure on page 8-154	Ensures that the objects in the replicated master group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by the advanced replication facility.
RESUME_MASTER_ACTIVITY procedure on page 8-155	Resumes normal replication activity after quiescing a replicated environment.
SUSPEND_MASTER_ACTIVITY procedure on page 8-160	Suspends replication activity for an object group
SWITCH_SNAPSHOT_MASTER procedure on page 8-161	Changes the master database of a snapshot replicated master group to another master site.
UNREGISTER_SNAPSHOT_REPGROUP procedure on page 8-162	Facilitates the administration of snapshots at their respective master sites by inserting/modifying/deleting from repcat$_repsite.
VALIDATE function on page 8-163	Validates the correctness of key conditions of a multiple master replication environment.
WAIT_MASTER_LOG procedure on page 8-166	Determines whether changes that were asynchronously propagated to a master site have been applied.

ADD_GROUPED_COLUMN procedure

This procedure adds members to an existing column group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 8–82 ADD_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table with which the column group is associated.
column_group	Name of the column group to which you are adding members.
list_of_column_names	Names of the columns that you are adding to the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. Use the single value '*' to create a column group that contains all of the columns in your table.

Table 8–83 ADD_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
missinggroup	Given column group does not exist.
missingcolumn	Given column does not exist in the designated table.
duplicatecolumn	Given column is already a member of another column group.
missingschema	Given schema does not exist.
notquiesced	Object group that the given table belongs to is not quiesced.

ADD_MASTER_DATABASE procedure

This procedure adds another master site to your replicated environment. This procedure regenerates all the triggers and their associated packages at existing master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_MASTER_DATABASE (
   gname                IN   VARCHAR2,
   master               IN   VARCHAR2,
   use_existing_objects IN   BOOLEAN  := TRUE,
   copy_rows            IN   BOOLEAN  := TRUE,
   comment              IN   VARCHAR2 := '',
   propagation_mode     IN   VARCHAR2 := 'ASYNCHRONOUS',
   fname                IN   VARCHAR2 := NULL);
```

Parameters

Table 8–84 ADD_MASTER_DATABASE Procedure Parameters

Parameter	Description
gname	Name of the object group being replicated. This object group must already exist at the master definition site.
master	Fully qualified database name of the new master database.
use_existing_objects	Indicate TRUE if you want to reuse any objects of the same type and shape that already exist in the schema at the new master site. See "Using Multimaster Replication" in the *Oracle8i Replication* manual for more information on how these changes are applied.
copy_rows	Indicate TRUE if you want the initial contents of a table at the new master site to match the contents of the table at the master definition site.
comment	This is added to the MASTER_COMMENT field of the RepSites view.
propagation_mode	Method of forwarding changes to and receiving changes from new master database. Accepted values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–85 ADD_MASTER_DATABASE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	replicated master group has not been suspended.
missingrepgroup	Object group does not exist at the given database site.
commfailure	New master is not accessible.
typefailure	An incorrect propagation mode was specified.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
duplrepgrp	Master site already exists.

ADD_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.

For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ADD_PRIORITY_datatype (
   gname              IN   VARCHAR2,
   pgroup             IN   VARCHAR2,
   value              IN   datatype,
   priority           IN   NUMBER);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 8–86 ADD_PRIORITY_datatype Procedure Parameters

Parameter	Description
`gname`	replicated master group for which you are creating a priority group.
`pgroup`	Name of the priority group.
`value`	Value of the priority group member. This would be one of the possible values of the associated `priority` column of a table using this priority group.
`priority`	Priority of this value. The higher the number, the higher the priority.

Exceptions

Table 8–87 ADD_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatevalue	Given value already exists in the priority group.
duplicatepriority	Given priority already exists in the priority group.
missingrepgroup	Given replicated master group does not exist.
missingprioritygroup	Given priority group does not exist.
typefailure	Given value has the incorrect datatype for the priority group.
notquiesced	Given replicated master group is not quiesced.

ADD_SITE_PRIORITY_SITE procedure

This procedure adds a new site to a site priority group. You must call this procedure from the master definition site.

For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
   gname          IN   VARCHAR2,
   name           IN   VARCHAR2
   site           IN   VARCHAR2,
   priority       IN   NUMBER);
```

Parameters

Table 8–88 ADD_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	replicated master group for which you are adding a site to a group.
name	Name of the site priority group to which you are adding a member.
site	Global database name of the site that you are adding.
priority	Priority level of the site that you are adding. A higher number indicates a higher priority level.

Exceptions

Table 8–89 ADD_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingpriority	Given site priority group does not exist.
duplicatepriority	Given priority level already exists for another site in the group.
duplicatevalue	Given site already exists in the site priority group.
notquiesced	replicated master group is not quiesced.

ADD_*conflicttype*_RESOLUTION procedure

This procedure designates 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.

Conflict Type	Procedure Name
update	ADD_UPDATE_RESOLUTION
uniqueness	ADD_UNIQUE_RESOLUTION
delete	ADD_DELETE_RESOLUTION

For more information about designating methods to resolve update conflicts, selecting uniqueness conflict resolution methods, and, assigning delete conflict resolution methods see "Conflict Resolution" in the *Oracle8i Replication* manual.

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,
   priority_group         IN   VARCHAR2     := NULL,
   function_name          IN   VARCHAR2     := NULL,
   comment                IN   VARCHAR2     := NULL);

DBMS_REPCAT.ADD_DELETE_RESOLUTION (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
   function_name          IN   VARCHAR2,
   comment                IN   VARCHAR2     := NULL
   method                 IN   VARCHAR2     := 'USER FUNCTION');
```

```
DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   constraint_name       IN   VARCHAR2,
   sequence_no           IN   NUMBER,
   method                IN   VARCHAR2,
   parameter_column_name IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
   function_name         IN   VARCHAR2     := NULL,
   comment               IN   VARCHAR2     := NULL);
```

Parameters

Table 8–90 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema containing the table to be replicated.
oname	Name of the table for which you are adding a conflict resolution routine.
column_group	Name of the column group for which you are adding a conflict resolution routine. Column groups are required for update conflict resolution routines only.
constraint_name	Name of the unique constraint or unique index for which you are adding a conflict resolution routine. Use the name of the unique index if it differs from the name of the associated unique constraint. Constraint names are required for uniqueness conflict resolution routines only.
sequence_no	Order in which the designated conflict resolution methods should be applied.
method	Type of conflict resolution routine that you want to create. This can be the name of one of the standard routines provided with advanced replication, or, if you have written your own routine, you should choose USER FUNCTION, and provide the name of your routine as the FUNCTION_NAME argument. The methods supported in this release are: MINIMUM, MAXIMUM, LATEST TIMESTAMP, EARLIEST TIMESTAMP, ADDITIVE, AVERAGE, PRIORITY GROUP, SITE PRIORITY, OVERWRITE, and DISCARD (for update conflicts) and APPEND SITE NAME, APPEND SEQUENCE NUMBER, and DISCARD (for uniqueness conflicts). There are no standard methods for delete conflicts.

Table 8–90 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
parameter_column_name	Name of the columns used to resolve the conflict. The standard methods operate on a single column. For example, if you are using the LATEST TIMESTAMP method for a column group, then you should pass the name of the column containing the timestamp value as this argument. If your are using a USER FUNCTION, then you can resolve the conflict using any number of columns. This argument accepts either a comma separated list of column names, or a PL/SQL table of type dbms_repcat.varchar2s. 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.
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 "Conflict Resolution" in the *Oracle8i Replication* manual. If you are using a different method, you can use the default value for this argument, NULL. This argument 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 argument, NULL.
comment	This user comment is added to the RepResolution view.

Exceptions

Table 8–91 ADD_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given object does not exist as a table in the given schema using row-level replication.
missingschema	Given schema does not exist.
missingcolumn	Column that you specified as part of the PARAMETER_COLUMN_NAME argument does not exist.
missinggroup	Given column group does not exist.
missingprioritygroup	priority group that you specified does not exist for the table.
invalidmethod	Resolution method that you specified is not recognized.
invalidparameter	Number of columns that you specified for the PARAMETER_COLUMN_NAME argument is invalid. (The standard routines take only one column name.)
missingfunction	User function that you specified does not exist.
missingconstraint	Constraint that you specified for a uniqueness conflict does not exist.
notquiesced	Object group that the given table belongs to is not quiesced.
duplicateresolution	Given conflict resolution method is already registered.
paramtype	Type is different from the type assigned to the priority group.

ALTER_MASTER_PROPAGATION procedure

This procedure alters the propagation method for a given object group at a given master site. This object group must be quiesced. You must call this procedure from the master definition site. If the master appears in the dblink_list or dblink_table, then ALTER_MASTER_PROPAGATION ignores that database link. You cannot change the propagation mode from a master to itself.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_PROPAGATION (
   gname               IN   VARCHAR2,
   master              IN   VARCHAR2,
   { dblink_list       IN   VARCHAR2,
   | dblink_table      IN   dbms_utility.dblink_array,}
   propagation_mode    IN   VARCHAR2 : ='asynchronous',
   comment             IN   VARCHAR2 := '');
```

> **Note:** This procedure is overloaded. The dblink_list and dblink_table parameters are mutually exclusive.

Parameters

Table 8–92 ALTER_MASTER_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the object group to which to alter the propagation mode.
master	Name of the master site at which to alter the propagation mode.
dblink_list	A comma-separated list of database links for which to alter propagation. If NULL, then all masters except the master site being altered are used by default.
dblink_table	A PL/SQL table, indexed from position 1, of database links for which to alter propagation.
propagation_mode	Determines the manner in which changes from the given master site are propagated to the sites identified by the list of database links. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS.
comment	This comment is added to the RepProp view.

Exceptions

Table 8–93 ALTER_MASTER_PROPAGATION Procedure Exceptions

Exception	Description
nonmasterdef	Local site is not the master definition site.
notquiesced	Local site is not quiesced.
typefailure	Propagation mode specified was not recognized.
nonmaster	List of database links includes a site that is not a master site.

ALTER_MASTER_REPOBJECT procedure

This procedure alters an object in your replicated environment. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   type         IN   VARCHAR2,
   ddl_text     IN   VARCHAR2,
   comment      IN   VARCHAR2     := '',
   retry        IN   BOOLEAN      := FALSE);
```

Parameters

Table 8–94 ALTER_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Schema containing the object that you want to alter.
oname	Name of the object that you want to alter.
type	Type of the object that you are altering. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY.
ddl_text	The DDL text that you want used to alter the object. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being altered.
comment	If not NULL, then this comment is added to the COMMENT field of the RepObject view.
retry	If retry is TRUE, then ALTER_MASTER_REPOBJECT alters the object only at masters whose object status is not VALID.

Exceptions

Table 8–95 ALTER_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Associated object group has not been suspended.
missingobject	Object identified by SNAME and ONAME does not exist.
typefailure	Given type parameter is not supported.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

Usage Notes

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.

ALTER_PRIORITY procedure

This procedure alters the priority level associated with a given priority group member. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY (
   gname            IN   VARCHAR2,
   pgroup           IN   VARCHAR2,
   old_priority     IN   NUMBER,
   new_priority     IN   NUMBER);
```

Parameters

Table 8–96 ALTER_PRIORITY Procedure Parameters

Parameter	Description
gname	replicated master group with which the priority group is associated.
pgroup	Name of the priority group containing the priority that you want to alter.
old_priority	Current priority level of the priority group member.
new_priority	New priority level that you want assigned to the priority group member.

Exceptions

Table 8–97 ALTER_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatepriority	New priority level already exists in the priority group.
missingrepgroup	Given replicated master group does not exist.
missingvalue	Value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_datatype.
missingprioritygroup	Given priority group does not exist.
notquiesced	Given replicated master group is not quiesced.

ALTER_PRIORITY_*datatype* procedure

This procedure alters the value of a member in a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your `priority` column.

For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY_datatype (
   gname        IN   VARCHAR2,
   pgroup       IN   VARCHAR2,
   old_value    IN   datatype,
   new_value    IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 8–98 ALTER_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	replicated master group with which the priority group is associated.
pgroup	Name of the priority group containing the value that you want to alter.
old_value	Current value of the priority group member.
new_value	New value that you want assigned to the priority group member.

Exceptions

Table 8–99 ALTER_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatevalue	New value already exists in the priority group.
missingrepgroup	Given replicated master group does not exist.
missingprioritygroup	Given priority group does not exist.
missingvalue	Old value does not exist.
paramtype	New value has the incorrect datatype for the priority group.
typefailure	Given value has the incorrect datatype for the priority group.
notquiesced	Given replicated master group is not quiesced.

ALTER_SITE_PRIORITY procedure

This procedure alters the priority level associated with a given site. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY (
   gname          IN   VARCHAR2,
   name           IN   VARCHAR2,
   old_priority   IN   NUMBER,
   new_priority   IN   NUMBER);
```

Parameters

Table 8–100 ALTER_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	replicated master group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_priority	Current priority level of the site whose priority level you want to change.
new_priority	New priority level for the site. A higher number indicates a higher priority level.

Exceptions

Table 8–101 ALTER_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingpriority	Old priority level is not associated with any group members.
duplicatepriority	New priority level already exists for another site in the group.
missingvalue	Old value does not already exist.
paramtype	New value has the incorrect datatype for the priority group.
notquiesced	replicated master group is not quiesced.

ALTER_SITE_PRIORITY_SITE procedure

This procedure alters the site associated with a given priority level. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE (
   gname     IN   VARCHAR2,
   name      IN   VARCHAR2,
   old_site  IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 8–102 ALTER_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	replicated master group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_site	Current global database name of the site to dissociate from the priority level.
new_site	New global database name that you want to associate with the current priority level.

Exceptions

Table 8–103 ALTER_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingpriority	Given site priority group does not exist.
missingvalue	Old site is not a group member.
notquiesced	replicated master group is not quiesced

ALTER_SNAPSHOT_PROPAGATION procedure

This procedure alters the propagation method for a given object group at the current snapshot site. This procedure pushes the deferred transaction queue at the snapshot site, locks the snapshot base tables, and regenerates any triggers and their associated packages. You must call this procedure from the snapshot site.

Syntax

```
DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION (
   gname                IN  VARCHAR2,
   propagation_mode     IN  VARCHAR2,
   comment              IN  VARCHAR2   := '');
```

Parameters

Table 8–104 ALTER_SNAPSHOT_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the object group for which to alter propagation mode.
propagation_mode	Manner in which changes from the current snapshot site are propagated to its associated master site. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS.
comment	This comment is added to the RepProp view.

Exceptions

Table 8–105 ALTER_SNAPSHOT_PROPAGATION Procedure Exceptions

Exception	Description
missingrepgroup	Given replicated master group does not exist.
typefailure	Propagation mode was specified incorrectly.
nonsnapshot	Current site is not a snapshot site for the given object group.
commfailure	Cannot contact master.

CANCEL_STATISTICS procedure

This procedure stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

Syntax

```
DBMS_REPCAT.CANCEL_STATISTICS (
   sname     IN   VARCHAR2,
   oname     IN   VARCHAR2);
```

Parameters

Table 8–106 CANCEL_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you do not want to gather conflict resolution statistics.

Exceptions

Table 8–107 CANCEL_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.
statnotreg	Given table is not currently registered to collect statistics.

COMMENT_ON_COLUMN_GROUP procedure

This procedure updates the comment field in the RepColumn_Group view for a column group. This comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP (
   sname           IN   VARCHAR2,
   oname           IN   VARCHAR2,
   column_group    IN   VARCHAR2,
   comment         IN   VARCHAR2);
```

Parameters

Table 8–108 COMMENT_ON_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the replicated table with which the column group is associated.
column_group	Name of the column group.
comment	Text of the updated comment that you want included in the GROUP_COMMENT field of the RepColumn_Group view.

Exceptions

Table 8–109 COMMENT_ON_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missinggroup	Given column group does not exist.
missingobj	Object is missing.

COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY procedure

COMMENT_ON_PRIORITY_GROUP updates the comment field in the 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 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 8–110 COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Parameters

Parameter	Description
gname	Name of the replicated master group.
pgroup/name	Name of the priority or site priority group.
comment	Text of the updated comment that you want included in the PRIORITY_COMMENT field of the RepPriority_Group view.

Exceptions

Table 8–111 COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Given replicated master group does not exist.
missingprioritygroup	Given priority group does not exist.

COMMENT_ON_REPGROUP procedure

This procedure updates the comment field in the REPGROUP view for a replicated master group. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPGROUP (
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 8–112 COMMENT_ON_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to comment on.
comment	Updated comment to include in the SCHEMA_COMMENT field of the RepGroup view.

Exceptions

Table 8–113 COMMENT_ON_REPGROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPSITES procedure

This procedure updates the comment field in the RepSite view for a replicated site. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPSITES (
   gname       IN   VARCHAR2,
   [ master    IN   VARCHAR,]
   comment     IN   VARCHAR2);
```

Parameters

Table 8–114 COMMENT_ON_REPSITES Procedure Parameters

Parameter	Description
gname	Name of the object group. This avoids confusion if a database is a master site in more than one replicated environment.
master	(Optional) The fully qualified database name of the master site that you want to comment on. To update comments at a snapshot site, omit this parameter.
comment	Text of the updated comment that you want to include in the MASTER_COMMENT field of the RepSites view.

Exceptions

Table 8–115 COMMENT_ON_REPSITES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPOBJECT procedure

This procedure updates the comment field in the RepObject view for a replicated object. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPOBJECT (
   sname    IN   VARCHAR2,
   oname    IN   VARCHAR2,
   type     IN   VARCHAR2,
   comment  IN   VARCHAR2);
```

Parameters

Table 8–116 COMMENT_ON_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to comment on.
type	Type of the object.
comment	Text of the updated comment that you want to include in the OBJECT_COMMENT field of the RepObject view.

Exceptions

Table 8–117 COMMENT_ON_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.
commfailure	At least one master site is not accessible.

COMMENT_ON_*conflicttype*_RESOLUTION procedure

This procedure updates the comment field in the 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.

Conflict Type	Procedure Name
update	COMMENT_ON_UPDATE_RESOLUTION
uniqueness	COMMENT_ON_UNIQUE_RESOLUTION
delete	COMMENT_ON_DELETE_RESOLUTION

The comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_UPDATE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   column_group         IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_UNIQUE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   constraint_name      IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_DELETE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);
```

Parameters

Table 8–118 COMMENT_ON_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema.
oname	Name of the replicated table with which the conflict resolution routine is associated.
column_group	Name of the column group with which the update conflict resolution routine is associated.
constraint_name	Name of the unique constraint with which the uniqueness conflict resolution routine is associated.
sequence_no	Sequence number of the conflict resolution procedure.
comment	The text of the updated comment that you want included in the RESOLUTION_COMMENT field of the RepResolution view.

Exceptions

Table 8–119 COMMENT_ON_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
missingresolution	Specified conflict resolution routine is not registered.

COMPARE_OLD_VALUES procedure

You have the option of comparing old column values for each non-key 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 and snapshot sites by invoking DBMS_REPCAT.COMPARE_OLD_VALUES at the master definition site.

Syntax

```
DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   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 8–120 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table.
column_list	A comma-separated list of the columns in the table. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_REPCAT.VARCHAR2S to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.
operation	Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete.

Table 8–120 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
compare	If compare is TRUE, the old values of the specified columns are compared when sent. If compare is FALSE, the old values of the specified columns are not compared when sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is TRUE for the table. The change takes effect at a master site or at a snapshot site the next time replication support is generated at that site with min_communication TRUE.

Note: The operation parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted or when non-key columns are 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.

Read "Minimizing Data Propagation for Update Conflict Resolution" in the *Oracle8i Replication* manual before changing the default behavior of Oracle.

Exceptions

Table 8–121 COMPARE_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	replicated master group has not been suspended.
typefailure	An illegal operation is given.

CREATE_MASTER_REPGROUP procedure

This procedure creates a new, empty, quiesced master replication object group.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPGROUP (
   gname            IN   VARCHAR2,
   group_comment    IN   VARCHAR2     := '',
   master_comment   IN   VARCHAR2     := ''),
   qualifier        IN   VARCHAR2     := '');
```

Parameters

Table 8–122 CREATE_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to create.
group_comment	This comment is added to the RepGroup view.
master_comment	This comment is added to the RepSites view.
qualifier	Connection qualifier for object group. Be sure to use the @ sign, as shown in the example: See "Managing Master Groups" in the *Oracle8i Replication* manual.

Exceptions

Table 8–123 CREATE_MASTER_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Object group already exists.
norepopt	Advanced replication option is not installed.
missingrepgroup	Object group name was not specified.
qualifiertoolong	Connection qualifier is too long.

CREATE_MASTER_REPOBJECT procedure

This procedure indicates that an object is a replicated object.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   type                 IN   VARCHAR2,
   use_existing_object  IN   BOOLEAN     := TRUE,
   ddl_text             IN   VARCHAR2    := NULL,
   comment              IN   VARCHAR2    := '',
   retry                IN   BOOLEAN     := FALSE
   copy_rows            IN   BOOLEAN     := TRUE,
   gname                IN   VARCHAR2    := '');
```

Parameters

Table 8–124 CREATE_MASTER_REPOBJECT Procedure Parameters

Parameters	Description
sname	Name of the schema in which the object that you want to replicate is located.
oname	Name of the object you are replicating. If DDL_TEXT is NULL, then this object must already exist in the given schema. To ensure uniqueness, table names should be a maximum of 27 bytes long, and package names should be no more than 24 bytes.
type	Type of the object that you are replicating. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY.
use_existing_object	Indicate TRUE if you want to reuse any objects of the same type and shape at the current master sites. See Table 8–126 for more information.
ddl_text	If the object does not already exist at the master definition site, then you must supply the DDL text necessary to create this object. PL/SQL packages, package bodies, procedures, and functions must have a trailing semicolon. SQL statements do not end with trailing semicolon. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being created.
comment	This comment is added to the OBJECT_COMMENT field of the RepObject view.
retry	Indicate TRUE if you want Oracle to reattempt to create an object that it was previously unable to create. Use this if the error was transient or has since been rectified; for example, if you previously had insufficient resources. If this is TRUE, then Oracle creates the object only at master sites whose object status is not VALID.
copy_rows	Indicate TRUE if you want the initial contents of a newly replicated object to match the contents of the object at the master definition site. See Table 8–126 for more information.
gname	Name of the object group in which you want to create the replicated object. The schema name is used as the default object group name if none is specified.

Table 8–125 CREATE_MASTER_REPOBJECT Procedure Exceptions

Exceptions	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	replicated master group has not been suspended.
duplicateobject	Given object already exists in the replicated master group and retry is FALSE, or if a name conflict occurs.
missingobject	Object identified by SNAME and ONAME does not exist and appropriate DDL has not been provided.
typefailure	Objects of the given type cannot be replicated.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.
notcompat	Not all remote masters in 7.3 compatibility mode.

Object Creations

Table 8–126 Object Creation at Master Sites

Object Already Exists?	COPY_ROWS	USE_EXISTING_OBJECTS	Result
yes	TRUE	TRUE	duplicatedobject message if objects do not match. For tables, use data from master definition site.
yes	FALSE	TRUE	duplicatedobject message if objects do not match. For tables, DBA must ensure contents are identical.
yes	TRUE/FALSE	FALSE	duplicatedobject message.
no	TRUE	TRUE/FALSE	Object is created. Tables populated using data from master definition site.
no	FALSE	TRUE/FALSE	Object is created. DBA must populate tables and ensure consistency of tables at all sites.

Usage Notes

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.

CREATE_SNAPSHOT_REPGROUP procedure

This procedure creates a new, empty snapshot group in your local database.

Syntax

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
   gname              IN   VARCHAR2,
   master             IN   VARCHAR2,
   comment            IN   VARCHAR2     := '',
   propagation_mode   IN   VARCHAR2     := 'ASYNCHRONOUS',
   fname              IN   VARCHAR2     := NULL);
```

Parameters

Table 8–127 CREATE_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group. This object group must exist at the given master site.
master	Fully qualified database name of the database in the replicated environment to use as the master.
comment	This comment is added to the RepGroup view.
propagation_mode	Method of propagation for all updatable snapshots in the object group. Acceptable values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–128 CREATE_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Object group already exists at the invocation site.
nonmaster	Given database is not a master site.
commfailure	Given database is not accessible.
norepopt	Advanced replication option is not installed.
typefailure	Propagation mode was specified incorrectly.
missingrepgroup	If replicated master group not at master site.

Usage Notes

CREATE_SNAPSHOT_REPGROUP automatically calls REGISTER_SNAPSHOT_REPGROUP, but ignores any errors that may have happened during registration.

CREATE_SNAPSHOT_REPOBJECT procedure

This procedure adds a replicated object to your snapshot site.

Syntax

```
DBMS_REPCAT.CREATE_SNAPSHOT_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);
```

Parameters

Table 8–129 CREATE_SNAPSHOT_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to add to the replicated snapshot object group. ONAME must exist at the associated master site.
type	Type of the object that you are replicating. The types supported for snapshot sites are: PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, SNAPSHOT, SYNONYM, TRIGGER, and VIEW.
ddl_text	For objects of type SNAPSHOT, the DDL text needed to create the object; for other types, use the default, (’’ (an empty string). If a snapshot with the same name already exists, then Oracle ignores the DDL and registers the existing snapshot as a replicated object. If the master table for a snapshot does not exist in the replicated master group of the master site designated for this schema, then Oracle raises a missingobject error.
comment	This comment is added to the OBJECT_COMMENT field of the RepObject view.
gname	Name of the replicated master group to which you are adding an object. The schema name is used as the default group name if none is specified.
gen_objs_owner	Name of the user you want to assign as owner of the transaction.

Table 8–129 CREATE_SNAPSHOT_REPOBJECT Procedure Parameters

Parameter	Description
min_communication	Set to FALSE if any 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, see "Conflict Resolution" in the *Oracle8i Replication* manual.
generate_80_compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 8–130 CREATE_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Master is no longer a master site.
missingobject	Given object does not exist in the master's replicated master group.
duplicateobject	Given object already exists with a different shape.
typefailure	Type is not an allowable type.
ddlfailure	DDL did not succeed.
commfailure	Master site is not accessible.
missingschema	Schema does not exist as a database schema.
badsnapddl	DDL was executed but snapshot does not exist.
onlyonesnap	Only one snapshot for master table can be created.
badsnapname	Snapshot base table differs from master table.
missingrepgroup	replicated master group does not exist.

Usage Notes

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.

DEFINE_COLUMN_GROUP procedure

This procedure creates an empty column group. You must call this procedure from the master definition site.

For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_COLUMN_GROUP (
   sname            IN   VARCHAR2,
   oname            IN   VARCHAR2,
   column_group     IN   VARCHAR2,
   comment          IN   VARCHAR2 := NULL);
```

Parameters

Table 8–131 DEFINE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a column group.
column_group	Name of the column group that you want to create.
comment	This user text is displayed in the RepColumn_Group view.

Exceptions

Table 8–132 DEFINE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
duplicategroup	Given column group already exists for the table.
notquiesced	Object group that the given table belongs to is not quiesced.

DEFINE_PRIORITY_GROUP procedure

This procedure creates a new priority group for a replicated master group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
   gname          IN   VARCHAR2,
   pgroup         IN   VARCHAR2,
   datatype       IN   VARCHAR2,
   fixed_length   IN   INTEGER := NULL,
   comment        IN   VARCHAR2 := NULL);
```

Parameters

Table 8–133 DEFINE_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	replicated master group for which you are creating a priority group.
pgroup	Name of the priority group that you are creating.
datatype	Datatype of the priority group members. The datatypes supported are: CHAR, VARCHAR2, NUMBER, DATE, RAW, NCHAR, and NVARCHAR2.
fixed_length	You must provide a column length for the CHAR datatype. All other types can use the default, NULL.
comment	This user comment is added to the RepPriority view.

Exceptions

Table 8–134 DEFINE_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
duplicatepriority group	Given priority group already exists in the replicated master group.
typefailure	Given datatype is not supported.
notquiesced	replicated master group is not quiesced.

DEFINE_SITE_PRIORITY procedure

This procedure creates a new site priority group for a replicated master group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY (
   gname        IN   VARCHAR2,
   name         IN   VARCHAR2,
   comment      IN   VARCHAR2 := NULL);
```

Parameters

Table 8–135 DEFINE_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	The replicated master group for which you are creating a site priority group.
name	Name of the site priority group that you are creating.
comment	This user comment is added to the `RepPriority` view.

Exceptions

Table 8–136 DEFINE_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
duplicate prioritygroup	Given site priority group already exists in the replicated master group.
notquiesced	replicated master group is not quiesced.

DO_DEFERRED_REPCAT_ADMIN procedure

This procedure executes the local outstanding deferred administrative procedures for the given replicated master group at the current master site, or (with assistance from job queues) for all master sites.

Syntax

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
   gname          IN   VARCHAR2,
   all_sites      IN   BOOLEAN := FALSE);
```

Parameters

Table 8–137 DO_DEFERRED_REPCAT_ADMIN Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
all_sites	If this is TRUE, then use a job to execute the local administrative procedures at each master.

Exceptions

Table 8–138 DO_DEFERRED_REPCAT_ADMIN Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible and all_sites is TRUE.

Usage Notes

DO_DEFERRED_REPCAT_ADMIN executes only those administrative requests submitted by the connected user that called DO_DEFERRED_REPCAT_ADMIN. Requests submitted by other users are ignored.

DROP_COLUMN_GROUP procedure

This procedure drops a column group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_COLUMN_GROUP (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   column_group IN   VARCHAR2);
```

Parameters

Table 8–139 DROP_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table whose column group you are dropping.
column_group	Name of the column group that you want to drop.

Exceptions

Table 8–140 DROP_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
referenced	Given column group is being used in conflict detection and resolution.
missingobject	Given table does not exist.
missinggroup	Given column group does not exist.
notquiesced	replicated master group that the table belongs to is not quiesced.

DROP_GROUPED_COLUMN procedure

This procedure removes members from a column group. You must call this procedure from the master definition site.

For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 8–141 DROP_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table in which the column group is located.
column_group	Name of the column group from which you are removing members.
list_of_column_names	Names of the columns that you are removing from the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s.

Exceptions

Table 8–142 DROP_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
notquiesced	replicated master group that the table belongs to is not quiesced.

DROP_MASTER_REPGROUP procedure

This procedure drops a replicated master group from your current site. To drop the replicated master group from all master sites, including the master definition site, you can call this procedure at the master definition site, and set the final argument to TRUE.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPGROUP (
   gname             IN VARCHAR2,
   drop_contents     IN BOOLEAN    := FALSE,
   all_sites         IN BOOLEAN     := FALSE);
```

Parameters

Table 8–143 DROP_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group that you want to drop from the current master site.
drop_contents	By default, when you drop the object 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 object 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 replicated master group are dropped from their associated schemas.
all_sites	If this is TRUE and if the invocation site is the master definition site, then the procedure synchronously multicasts the request to all masters. In this case, execution is immediate at the master definition site and may be deferred at all other master sites.

Exceptions

Table 8–144 DROP_MASTER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
nonmasterdef	Invocation site is not the master definition site and ALL_SITES is TRUE.
commfailure	At least one master site is not accessible and ALL_SITES is TRUE.
fullqueue	Deferred RPC queue has entries for the replicated master group.
masternotremoved	Master does not recognize the masterdef and ALL_SITES is TRUE.

DROP_MASTER_REPOBJECT procedure

This procedure drops a replicated object from a replicated master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN     := FALSE);
```

Parameters

Table 8–145 DROP_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to remove from the replicated master group.
type	Type of object that you want to drop.
drop_objects	By default, the object remains in the schema, but is dropped from the replicated master group; that is, any changes to the object are no longer replicated to other master and snapshot sites. To completely remove the object from the replicated environment, set this argument to TRUE.

Exceptions

Table 8–146 DROP_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.
commfailure	At least one master site is not accessible.

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 "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY(
      gname          IN   VARCHAR2,
      pgroup         IN   VARCHAR2,
      priority_num   IN   NUMBER);
```

Parameters

Table 8–147 DROP_PRIORITY Procedure Parameters

Parameter	Description
gname	replicated master group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
priority_num	Priority level of the priority group member that you want to remove from the group.

Exceptions

Table 8–148 DROP_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingprioritygroup	Given priority group does not exist.
notquiesced	replicated master group is not quiesced.

DROP_PRIORITY_GROUP procedure

This procedure drops a priority group for a given replicated master group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_GROUP (
   gname      IN   VARCHAR2,
   pgroup     IN   VARCHAR2);
```

Parameters

Table 8–149 DROP_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	replicated master group with which the priority group is associated.
pgroup	Name of the priority group that you want to drop.

Exceptions

Table 8–150 DROP_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
referenced	Given priority group is being used in conflict resolution.
notquiesced	Given replicated master group is not quiesced.

DROP_PRIORITY_*datatype* procedure

This procedure drops a member of a priority group by value. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your `priority` column.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_datatype (
   gname    IN   VARCHAR2,
   pgroup   IN   VARCHAR2,
   value    IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 8–151 DROP_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	replicated master group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
value	Value of the priority group member that you want to remove from the group.

Exceptions

Table 8–152 DROP_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingprioritygroup	Given priority group does not exist.
paramtype, typefailure	Value has the incorrect datatype for the priority group.
notquiesced	Given replicated master group is not quiesced

DROP_SITE_PRIORITY procedure

This procedure drops a site priority group for a given replicated master group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2);
```

Parameters

Table 8–153 DROP_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	replicated master group with which the site priority group is associated.
name	Name of the site priority group that you want to drop.

Exceptions

Table 8–154 DROP_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
referenced	Given site priority group is being used in conflict resolution.
notquiesced	Given replicated master group is not quiesced

DROP_SITE_PRIORITY_SITE procedure

This procedure drops a given site, by name, from a site priority group. You must call this procedure from the master definition site.

See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2,
   site        IN   VARCHAR2);
```

Parameters

Table 8–155 DROP_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	replicated master group with which the site priority group is associated.
name	Name of the site priority group whose member you are dropping.
site	Global database name of the site you are removing from the group.

Exceptions

Table 8–156 DROP_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated master group does not exist.
missingpriority	Given site priority group does not exist.
notquiesced	Given replicated master group is not quiesced.

DROP_SNAPSHOT_REPGROUP procedure

This procedure drops a snapshot site from your replicated environment.

Syntax

```
DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   gname                IN   VARCHAR2,
   drop_contents        IN   BOOLEAN   := FALSE);
```

Parameters

Table 8–157 DROP_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group that you want to drop from the current snapshot site. All objects generated to support replication, such as triggers and packages, are dropped.
drop_contents	By default, when you drop the replicated master group at a snapshot 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 replicated master group are dropped from their schemas.

Exceptions

Table 8–158 DROP_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingrepgroup	Specified object group does not exist.

Usage Notes

DROP_SNAPSHOT_REPGROUP automatically calls UNREGISTER_SNAPSHOT_REPGROUP to unregister the snapshot, but ignores any errors that may have occurred during unregistration.

DROP_SNAPSHOT_REPOBJECT procedure

This procedure drops a replicated object from a snapshot site.

Syntax

```
DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN  := FALSE);
```

Parameters

Table 8–159 DROP_SNAPSHOT_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to drop from the replicated master group.
type	Type of the object that you want to drop.
drop_objects	By default, the object remains in its associated schema, but is dropped from its associated object group. To completely remove the object from its schema at the current snapshot site, set this argument to TRUE.

Exceptions

Table 8–160 DROP_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.

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 Type	Procedure Name
update	DROP_UPDATE_RESOLUTION
uniqueness	DROP_UNIQUE_RESOLUTION
delete	DROP_DELETE_RESOLUTION

Syntax

```
DBMS_REPCAT.DROP_UPDATE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   column_group        IN   VARCHAR2,
   sequence_no         IN   NUMBER);

DBMS_REPCAT.DROP_DELETE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   sequence_no         IN   NUMBER);

DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   constraint_name     IN   VARCHAR2,
   sequence_no         IN   NUMBER);
```

Parameters

Table 8–161 DROP_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table for which you want to drop a conflict resolution routine.
column_group	Name of the column group for which you want to drop an update conflict resolution routine.
constraint_name	Name of the Unique constraint for which you want to drop a unique conflict resolution routine.
sequence_no	Sequence number assigned to the conflict resolution method that you want to drop. This number uniquely identifies the routine.

Exceptions

Table 8–162 DROP_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given object does not exist as a table in the given schema, or a conflict resolution routine with the given sequence number is not registered.
notquiesced	replicated master group is not quiesced.

EXECUTE_DDL procedure

This procedure supplies DDL that you want to have executed at some or all master sites. You can call this procedure only from the master definition site.

Syntax

```
DBMS_REPCAT.EXECUTE_DDL (
   gname           IN   VARCHAR2,
   { master_list   IN   VARCHAR2     := NULL,
    | master_table  IN   DBMS_UTILITY.DBLINK_ARRAY,}
   DDL_TEXT        IN   VARCHAR2);
```

Parameters

Table 8–163 EXECUTE_DDL Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
master_list	A comma-separated list of master sites at which you want to execute the supplied DDL. There must be no extra white space between site names. The default value, NULL, indicates that the DDL should be executed at all sites, including the master definition site.
master_table	A table of master sites at which you want to execute the supplied DDL. The first master should be at offset 1, the second at offset 2, and so on.
ddl_text	The DDL that you want to have executed at each of the given master sites.

Exceptions

Table 8–164 EXECUTE_DDL Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one site is not a master site.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

Usage Notes

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. This procedure is overloaded. The MASTER_LIST and MASTER_TABLE parameters are mutually exclusive.

GENERATE_REPLICATION_SUPPORT procedure

This procedure generates the triggers and packages needed to support replication. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
  sname                    IN    VARCHAR2,
  oname                    IN    VARCHAR2,
  type                     IN    VARCHAR2,
  package_prefix           IN    VARCHAR2   := NULL,
  procedure_prefix         IN    VARCHAR2   := NULL,
  distributed              IN    BOOLEAN    := TRUE,
  gen_objs_owner           IN    VARCHAR2   := NULL,
  min_communication        IN    BOOLEAN    := TRUE,
  generate_80_compatible   IN    BOOLEAN    := TRUE);
```

Parameters

Table 8–165 GENERATE_REPLICATION_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	Name of the object for which you are generating replication support.
type	Type of the object. The types supported are: TABLE, PACKAGE, and PACKAGE BODY.
package_prefix	For objects of type PACKAGE or PACKAGE BODY this value is prepended to the generated wrapper package name. The default is DEFER_.
procedure_prefix	For objects of type PACKAGE or PACKAGE BODY, this value is prepended to the generated wrapper procedure names. By default, no prefix is assigned.
distributed	This must be set to TRUE.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, the objects will be created in SNAME.

Table 8–165 GENERATE_REPLICATION_SUPPORT Procedure Parameters

Parameter	Description
min_communication	Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE when you want propagation of new and old values to be minimized. The default is TRUE. For more information, see "Conflict Resolution' in the *Oracle8i Replication* manual.
generate_80_compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 8–166 GENERATE_REPLICATION_SUPPORT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information or as a package (body) awaiting wrapper generation.
typefailure	Given type parameter is not supported.
notquiesced	replicated master group has not been suspended.
commfailure	At least one master site is not accessible.
missingschema	Schema does not exist.
dbnotcompatible	One of the masters is not 7.3 compatible.
duplicateobject	Object already exists.

GENERATE_SNAPSHOT_SUPPORT procedure

This procedure activates triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.You must call this procedure from the snapshot site.

Syntax

```
DBMS_REPCAT.GENERATE_SNAPSHOT_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 8–167 GENERATE_SNAPSHOT_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	The name of the object that you are generating support for.
type	Type of the object. The types supported are SNAPSHOT, PACKAGE, and PACKAGE BODY.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, the objects will be created in SNAME.
min_communication	If TRUE, then the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group.
generate_80_ compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 8–168 GENERATE_SNAPSHOT_SUPPORT Procedure Exceptions

Exceptions	Descriptions
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Given object does not exist as a snapshot in the replicated schema awaiting row/column-level replication information or as a package (body) awaiting wrapper generation.
typefailure	Given type parameter is not supported.
missingschema	Specified owner of generated objects does not exist.
missingremoteobject	Master object has not yet generated replication support.
commfailure	Master is not accessible.

Usage Notes

CREATE_SNAPSHOT_REPOBJECT automatically generates snapshot support for updatable snapshots.

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.

For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.MAKE_COLUMN_GROUP (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 8–169 MAKE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a new column group.
column_group	Name that you want assigned to the column group that you are creating.
list_of_column_names	Names of the columns that you are grouping. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. Use the single value '*' to create a column group that contains all of the columns in your table.

Exceptions

Table 8–170 MAKE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicategroup	Given column group already exists for the table.
missingobject	Given table does not exist.
missingcolumn	Given column does not exist in the designated table.
duplicatecolumn	Given column is already a member of another column group.
notquiesced	replicated master group is not quiesced.

PURGE_MASTER_LOG procedure

This procedure removes local messages in the RepCatLog associated with a given identification number, source, or replicated master group.

Syntax

```
DBMS_REPCAT.PURGE_MASTER_LOG (
   id      IN   NATURAL,
   source  IN   VARCHAR2,
   gname   IN   VARCHAR2);
```

Parameters

Table 8–171 PURGE_MASTER_LOG Procedure Parameters

Parameter	Description
id	Identification number of the request, as it appears in the RepCatLog view.
source	Master site from which the request originated.
gname	Name of the replicated master group for which the request was made.

Exceptions

Table 8–172 PURGE_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	gname is not NULL, and the invocation site is not a master site.

PURGE_STATISTICS procedure

This procedure removes information from the RepResolution_Statistics view.

Syntax

```
DBMS_REPCAT.PURGE_STATISTICS (
   sname      IN   VARCHAR2,
   oname      IN   VARCHAR2,
   start_date IN   DATE,
   end_date   IN   DATE);
```

Parameters

Table 8–173 PURGE_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the replicated table is located.
oname	Name of the table whose conflict resolution statistics you want to purge.
start_date/end_date	Range of dates for which you want to purge statistics. If START_DATE is NULL, then purge all statistics up to the END_DATE. If END_DATE is NULL, then purge all statistics after the START_DATE.

Exceptions

Table 8–174 PURGE_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.
statnotreg	Table not registered to collect statistics.

REFRESH_SNAPSHOT_REPGROUP procedure

This procedure refreshes a snapshot site object group with the most recent data from its associated master site.

Syntax

```
DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP (
   gname                 IN   VARCHAR2,
   drop_missing_contents IN   BOOLEAN    := FALSE,
   refresh_snapshots     IN   BOOLEAN    := FALSE,
   refresh_other_objects IN   BOOLEAN    := FALSE);
```

Parameters

Table 8–175 REFRESH_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
drop_missing_ contents	If an object was dropped from the replicated master group, then it is not automatically dropped from the schema at the snapshot site. It is simply no longer replicated; that is, changes to this object are no longer sent to its associated master site. Snapshots can continue to be refreshed from their associated master tables; however, any changes to an updatable snapshot are lost. When an object is dropped from the object group, you can choose to have it dropped from the schema entirely by setting this argument to TRUE.
refresh_snapshots	Set this to TRUE to refresh the contents of the snapshots in the replicated master group.
refresh_other_ objects	Set this to TRUE to refresh the contents of the non-snapshot objects in the replicated master group.

Exceptions

Table 8–176 REFRESH_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Master is no longer a master site.
commfailure	Master is not accessible.
missingrepgroup	Object group name not specified.

REGISTER_SNAPSHOT_REPGROUP procedure

This procedure facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from `registered_snapshot_groups`.

Syntax

```
DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP (
   gname            IN   VARCHAR2,
   snapsite         IN   VARCHAR2,
   comment          IN   VARCHAR2  := NULL,
   rep_type         IN   NUMBER    := reg_unknown,
   fname            IN   VARCHAR2  := NULL);
```

Parameters

Table 8–177 REGISTER_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group to be registered.
snapsite	Global name of the snapshot site.
comment	Comment for the snapshot site or update for an existing comment.
rep_type	Version of the snapshot group. Valid constants that can be assigned include `reg_uknown` (the default), `reg_v7_group`, `reg_v8_group`, and `reg_repapi_group`.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–178 REGISTER_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
missingrepgroup	Object group name not specified.
nullsitename	A snapshot site was not specified.
nonmaster	Procedure must be executed at the snapshot's master site.
duplicaterepgroup	Object already exists.

REGISTER_STATISTICS procedure

This procedure collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.

Syntax

```
DBMS_REPCAT.REGISTER_STATISTICS (
   sname IN   VARCHAR2,
   oname IN   VARCHAR2);
```

Parameters

Table 8–179 REGISTER_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you want to gather conflict resolution statistics.

Exceptions

Table 8–180 REGISTER_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.

RELOCATE_MASTERDEF procedure

This procedure changes your master definition site to another master site in your replicated environment.

Syntax

```
DBMS_REPCAT.RELOCATE_MASTERDEF (
   gname                 IN   VARCHAR2,
   old_masterdef         IN   VARCHAR2,
   new_masterdef         IN   VARCHAR2,
   notify_masters        IN   BOOLEAN    := TRUE,
   include_old_masterdef IN   BOOLEAN    := TRUE,
   require_flavor_change IN   BOOLEAN    := FALSE);
```

Parameters

Table 8–181 RELOCATE_MASTERDEF Procedure Parameters

Parameter	Description
gname	Name of the object group whose master definition you want to relocate.
old_masterdef	Fully qualified database name of the current master definition site.
new_masterdef	Fully qualified database name of the existing master site that you want to make the new master definition site.
notify_masters	If this is TRUE, then the procedure synchronously multicasts the change to all masters (including OLD_MASTERDEF only if INCLUDE_OLD_MASTERDEF is TRUE). If any master does not make the change, then roll back the changes at all masters.
include_old_ masterdef	If NOTIFY_MASTERS is TRUE and if INCLUDE_OLD_MASTERDEF is also TRUE, then the old master definition site is also notified of the change.
require_flavor_ change	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 8–182 RELOCATE_MASTERDEF Procedure Exceptions

Exception	Description
nonmaster	NEW_MASTERDEF is not a master site or the invocation site is not a master site.
nonmasterdef	OLD_MASTERDEF is not the master definition site.
commfailure	At least one master site is not accessible and NOTIFY_MASTERS is TRUE.

Usage Notes

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 TRUE and INCLUDE_OLD_MASTERDEF TRUE.

If just the master definition site fails, then you should invoke RELOCATE_MASTERDEF with NOTIFY_MASTERS TRUE and INCLUDE_OLD_MASTERDEF 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 FALSE.

REMOVE_MASTER_DATABASES procedure

This procedure removes one or more master databases from a replicated environment. This procedure regenerates the triggers and their associated packages at the remaining master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.REMOVE_MASTER_DATABASES (
   gname           IN   VARCHAR2,
   master_list     IN   VARCHAR2 |
   master_table    IN   DBMS_UTILITY.DBLINK_ARRAY);
```

Parameters

Table 8–183 REMOVE_MASTER_DATABASES Procedure Parameters

Parameter	Description
gname	Name of the object group associated with the replicated environment. This prevents confusion if a master database is involved in more than one replicated environment.
master_list	A comma-separated list of fully qualified master database names that you want to remove from the replicated environment. There must be no white space between names in the list.
master_table	In place of a list, you may also specify the database names in a PL/SQL table of type DBMS_UTILITY.DBLINK_ARRAY.

Exceptions

Table 8–184 REMOVE_MASTER_DATABASES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one of the given databases is not a master site.
reconfigerror	One of the given databases is the master definition site.
commfailure	At least one remaining master site is not accessible.

REPCAT_IMPORT_CHECK procedure

This procedure ensures that the objects in the replicated 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.

Syntax

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK (
   gname      IN   VARCHAR2,
   master     IN   BOOLEAN);
```

Parameters

Table 8–185 REPCAT_IMPORT_CHECK Procedure Parameters

Parameter	Description
gname	Name of the replicated master group. If you omit both parameters, then the procedure checks all replicated master groups at your current site.
master	Set this to TRUE if you are checking a master site or FALSE if you are checking a snapshot site.

Exceptions

Table 8–186 REPCAT_IMPORT_CHECK Procedure Exceptions

Exception	Description
nonmaster	MASTER is TRUE and either the database is not a master site for the object group or the database is not the expected database.
nonsnapshot	MASTER is FALSE and the database is not a snapshot site for the object group.
missingobject	A valid replicated object in the object group does not exist.
missingrepgroup	The given group name does not exist.

RESUME_MASTER_ACTIVITY procedure

This procedure resumes normal replication activity after quiescing a replicated environment.

Syntax

```
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
   gname       IN  VARCHAR2,
   override    IN  BOOLEAN := FALSE);
```

Parameters

Table 8–187 RESUME_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
override	If this is TRUE, then it ignores any pending RepCat administration 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 administration request for gname at that master.

Exceptions

Table 8–188 RESUME_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	replicated master group is not quiescing or quiesced.
commfailure	At least one master site is not accessible.

SEND_OLD_VALUES procedure

You have the option of sending old column values for each non-key column of a replicated table for updates and deletes. The default is to send old values for all columns. You can change this behavior at all master and snapshot sites by invoking DBMS_REPCAT.SEND_OLD_VALUES at the master definition site.

Syntax

```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   operation       IN  VARCHAR2 := 'UPDATE',
   send            IN  BOOLEAN := TRUE );
```

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

Parameters

Table 8–189 SEND_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table.
column_list	A comma-separated list of the columns in the table. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_REPCAT.VARCHAR2S to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.
operation	Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete.

Table 8–189 SEND_OLD_VALUES Procedure Parameters

Parameter	Description
send	If TRUE, then the old values of the specified columns are sent. If FALSE, then the old values of the specified columns are not sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is TRUE for the table. The change takes effect at a master site or at a snapshot site the next time replication support is generated at that site with min_communication TRUE.

Note: The operation parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted or when non-key columns are 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.

Read "Minimizing Data Propagation for Update Conflict Resolution" in the *Oracle8i Replication* manual before changing the default behavior of Oracle.

Exceptions

Table 8–190 SEND_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	replicated master group has not been suspended.
typefailure	An illegal operation is given.

SET_COLUMNS procedure

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.

See "Using Multimaster Replication" in the *Oracle8i Replication* manual

Syntax

```
DBMS_REPCAT.SET_COLUMNS (
   sname           IN    VARCHAR2,
   oname           IN    VARCHAR2,
   { column_list   IN    VARCHAR2
   | column_table  IN    DBMS_UTILITY.NAME_ARRAY } );
```

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

Parameters

Table 8–191 SET_COLUMNS Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table.
column_list	A comma-separated list of the columns in the table that you want to use as a primary key. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_UTILITY.NAME_ARRAY to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.

Exceptions

Table 8–192 SET_COLUMNS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.

SUSPEND_MASTER_ACTIVITY procedure

This procedure suspends replication activity for an object group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
   gname    IN    VARCHAR2);
```

Parameters

Table 8–193 SUSPEND_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the object group for which you want to suspend activity.

Exceptions

Table 8–194 SUSPEND_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notnormal	replicated master group is not in normal operation.
commfailure	At least one master site is not accessible.

Usage Notes

The current implementation of SUSPEND_MASTER_ACTIVITY quiesces all replicated master groups at each master site.

SWITCH_SNAPSHOT_MASTER procedure

This procedure changes the master database of a snapshot replicated master group to another master site. This procedure does a full refresh of the affected snapshots and regenerates the triggers and their associated packages as needed. This procedure does not push the queue to the old master site before changing masters.

Syntax

```
DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER (
   gname                IN   VARCHAR2,
   master               IN   VARCHAR2);
```

Parameters

Table 8–195 SWITCH_SNAPSHOT_MASTER Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group for which you want to change master sites.
master	Fully qualified database name of the new master database to use for the snapshot site.

Exceptions

Table 8–196 SWITCH_SNAPSHOT_MASTER Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Given database is not a master site.
commfailure	Given database is not accessible.

UNREGISTER_SNAPSHOT_REPGROUP procedure

This procedure facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from `registered_snapshot_groups`.

Syntax

```
DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP (
   gname     IN   VARCHAR2,
   snapsite  IN   VARCHAR2);
```

Parameters

Table 8–197 UNREGISTER_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group to be unregistered.
snapsite	Global name of the snapshot site.

VALIDATE function

This function validates the correctness of key conditions of a multiple master replication environment. This is overloaded.

Syntax

```
DBMS_REPCAT.VALIDATE (
   gname               IN  VARCHAR2,
   check_genflags      IN  BOOLEAN := FALSE,
   check_valid_objs    IN  BOOLEAN := FALSE,
   check_links_sched   IN  BOOLEAN := FALSE,
   check_links         IN  BOOLEAN := FALSE,
   error_table         OUT dbms_repcat.validate_err_table )
  RETURN BINARY_INTEGER;

DBMS_REPCAT.VALIDATE (
   gname               IN  VARCHAR2,
   check_genflags      IN  BOOLEAN := FALSE,
   check_valid_objs    IN  BOOLEAN := FALSE,
   check_links_sched   IN  BOOLEAN := FALSE,
   check_links         IN  BOOLEAN := FALSE,
   error_msg_table     OUT DBMS_UTILITY.UNCL_ARRAY,
   error_num_table     OUT DBMS_UTILITY.NUMBER_ARRAY )
  RETURN BINARY_INTEGER;
```

Parameters

Table 8–198 VALIDATE Function Parameters

Parameter	Description
gname	Name of the master group to validate.
check_genflags	Check whether all the objects in the group are generated. This must be done at the masterdef site only.
check_valid_objs	Check that the underlying objects for objects in the group valid. This must be done at the masterdef site only. The masterdef site goes to all other sites and checks that the underlying objects are valid. The validity of the objects is checked within the schema of the connected user.
check_links_sched	Check whether the links are scheduled for execution. This should be invoked at each master site.

Table 8–198 VALIDATE Function Parameters

Parameter	Description
check_links	Check whether the connected user (repadmin), as well as the propagator, have correct links for replication to work properly. Checks that the links exist in the database and are accessible. This should be invoked at each master site.
error_table	Returns the message and numbers of all errors that are found.
error_msg_table	Returns the messages of all errors that are found.
error_num_table	Returns the numbers of all errors that are found.

Exceptions

Table 8–199 VALIDATE Function Exceptions

Exception	Description
missingdblink	Database link does not exist in the schema of the replication propagator or has not been scheduled. Ensure that the database link exists in the database, is accessible, and is scheduled for execution.
dblinkmismatch	Database link name at the local node does not match the global name of the database that the link accesses. Ensure that global names is set to true and the link name matches the global name.
dblinkuidmismatch	User name of the replication administration user at the local node and the user name at the node corresponding to the database link are not the same. Advanced replication expects the two users to be the same. Ensure that the user ID of the replication administration user at the local node and the user ID at the node corresponding to the database link are the same.
objectnotgenerated	Object has not been generated at other master sites or is still being generated. Ensure that the object is generated by calling generate_replication_support and do_deferred_repcat_admin for the object at the masterdef site.
opnotsupported	Operation is not supported if the object group is replicated at a pre-V8 node. Ensure that all nodes of the replicated master group are V8.

Usage Notes

The return value of VALIDATE is the number of errors found. The function's OUT parameter(s) returns any errors that are found. In the first interface function, the

ERROR_TABLE consists of an array of records. Each record has a VARCHAR2 and a NUMBER in it. The string field contains the error message and the number field contains the Oracle error number.

The second interface is similar except that there are two OUT arrays. A VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

WAIT_MASTER_LOG procedure

This procedure determines whether changes that were asynchronously propagated to a master site have been applied.

Syntax

```
DBMS_REPCAT.WAIT_MASTER_LOG (
   gname          IN    VARCHAR2,
   record_count   IN    NATURAL,
   timeout        IN    NATURAL,
   true_count     OUT   NATURAL);
```

Parameters

Table 8–200 WAIT_MASTER_LOG Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
record_count	Procedure returns whenever the number of incomplete activities is at or below this threshold.
timeout	Maximum number of seconds to wait before the procedure returns.
true_count (out parameter)	Returns the number of incomplete activities.

Exceptions

Table 8–201 WAIT_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.

DBMS_REPCAT_ADMIN Package

Summary of Subprograms

Table 8–202 DBMS_REPCAT_ADMIN Package Subprograms

Subprogram	Description
GRANT_ADMIN_ANY_SCHEMA procedure on page 8-168	Grants the necessary privileges to the replication administrator to administer any replicated master group at the current site.
GRANT_ADMIN_SCHEMA procedure on page 8-169	Grants the necessary privileges to the replication administrator to administer a schema at the current site.
REGISTER_USER_REPGROUP procedure on page 170	Assigns proxy snapshot administrator or receiver privileges at the master site for use with remote sites.
REVOKE_ADMIN_ANY_SCHEMA procedure on page 8-172	Revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_ANY_SCHEMA.
REVOKE_ADMIN_SCHEMA procedure on page 8-173	Revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_SCHEMA.
UNREGISTER_USER_REPGROUP procedure on page 8-174	Revokes the privileges and roles from the proxy snapshot administrator or receiver that would be granted by the REGISTER_USER_REPGROUP procedure.

GRANT_ADMIN_ANY_SCHEMA procedure

This procedure grants the necessary privileges to the replication administrator to administer any replicated master group at the current site.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
   username  IN   VARCHAR2);
```

Parameters

Table 8–203 GRANT_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator to whom you want to grant the necessary privileges and roles to administer any replicated master groups at the current site.

Exceptions

Table 8–204 GRANT_ADMIN_ANY_REPGROUP Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

GRANT_ADMIN_SCHEMA procedure

This procedure grants the necessary privileges to the replication administrator to administer a schema at the current site. This procedure is most useful if your object group does not span schemas.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 8–205 GRANT_ADMIN_REPSCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator. This user is then granted the necessary privileges and roles to administer the schema of the same name within a replicated master group at the current site.

Exceptions

Table 8–206 GRANT_ADMIN_REPSCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REGISTER_USER_REPGROUP procedure

This procedure assigns proxy snapshot administrator or receiver privileges at the master site for use with remote sites. This procedure grants only the necessary privileges to the proxy snapshot administrator or receiver, avoiding having to grant the powerful privileges granted by the GRANT_ADMIN_SCHEMA or GRANT_ADMIN_ANY_SCHEMA procedures.

See "Advanced Techniques" in the *Oracle8i Replication* manual for more information on trusted versus untrusted security models.

Syntax

```
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
   username         IN   VARCHAR2,
   privilege_type   IN   VARCHAR2,
   list_of_gnames   IN   VARCHAR2 |
   table_of_gnames  IN   dbms_utility.name_array);
```

Parameters

Table 8–207 REGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are giving either proxy snapshot administrator or receiver privileges to.
privilege_type	Specifies the privilege type you are assigning. Use the following values for to define your privilege_type: RECEIVER for receiver privileges PROXY_SNAPADMIN for proxy snapadmin privileges.
list_of_gnames	Comma-separated list of object groups you want a user registered for receiver privileges. There must be no whitespace between entries in the list. If you set list_of_gnames to NULL, then the user is registered for all object groups, even object 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 object group in the list causes registration to fail for the entire list. If you specify a value for list_of_gnames, then do not specify a value for table_of_gnames.
table_of_gnames	PL/SQL table of object groups you want a user registered for receiver privileges. The PL/SQL table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based. Use the single value NULL to register the user for all object groups. An invalid object group in the table causes registration to fail for the entire table. If you specify a value for table_of_gnames, then do not specify a value for list_of_gnames.

Exceptions

Table 8–208 REGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified object group does not exist or the invocation database is not a master.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

REVOKE_ADMIN_ANY_SCHEMA procedure

This procedure revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_ANY_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_ANY_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 8–209 REVOKE_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 8–210 REVOKE_ADMIN_ANY_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REVOKE_ADMIN_SCHEMA procedure

This procedure revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 8–211 REVOKE_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 8–212 REVOKE_ADMIN_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

UNREGISTER_USER_REPGROUP procedure

This procedure revokes the privileges and roles from the proxy snapshot administrator or receiver that would be 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);
```

Parameters

Table 8–213 UNREGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are unregistering.
privilege_type	Specifies the privilege type you are revoking. Use the following values for to define your privilege_type: RECEIVER for receiver privileges PROXY_SNAPADMIN for proxy snapadmin privileges.
list_of_gnames	Comma-separated list of object groups you want a user unregistered for receiver privileges. There must be no whitespace between entries in the list. If you set list_of_gnames to NULL, then the user is unregistered for all object groups registered. You must use named notation in order to set list_of_gnames to NULL. An invalid object group in the list causes unregistration to fail for the entire list. If you specify a value for list_of_gnames, then do not specify a value for table_of_gnames.
table_of_gnames	PL/SQL table of object groups you want a user unregistered for receiver privileges. The PL/SQL table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based. Use the single value NULL to unregister the user for all object groups registered. An invalid object group in the table causes unregistration to fail for the entire table. If you specify a value for table_of_gnames, then do not specify a value for list_of_gnames.

Exceptions

Table 8–214 UNREGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified object group does not exist or the invocation database is not a master.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

DBMS_REPCAT_INSTANTIATE Package

Summary of Subprograms

Table 8–215 DBMS_REPCAT_INSTANTIATE Package Subprograms

Subprogram	Description
DROP_SITE_INSTANTIATION procedure on page 8-177	Public procedure that removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
INSTANTIATE_OFFLINE procedure on page 8-178	Public procedure that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.
INSTANTIATE_ONLINE procedure on page 8-180	Public procedure that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online.

DROP_SITE_INSTANTIATION procedure

This procedure drops a template instantiation at a target site. This procedure removes all related meta data at the master site and disables the specified site from refreshing their snapshots. You need to execute this procedure as the user that originally instantiated the template (to see who instantiated the template, query the REPCAT_TEMPLATE_SITES View, which is described on page 9-11.

Syntax

The parameter for the DROP_SITE_INSTANTIATION procedure is described in Table 8–216:

```
DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
     refresh_template_name  IN   VARCHAR2,
     site_name              IN   VARCHAR2,
     repapi_site_id         IN   NUMBER := -1e-130)
```

Table 8–216 Parameter for DROP_SITE_INSTANTIATION

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
site_name	Identifies the Oracle server site where you want to drop the specified template instantiation (if you specify a SITE_NAME, do not specify a REPAPI_SITE_ID).
repapi_site_id	Identifies the REPAPI location where you want to drop the specified template instantiation (if you specify a REPAPI_ SITE_ID, do not specify a SITE_NAME).

INSTANTIATE_OFFLINE procedure

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline. This generated script should be used at remote snapshot sites that are NOT able to remain connected to the master site for an extended amount of time. This is an ideal solution where the remote snapshot site is a laptop. Use the packaging tool in Replication Manager to package the generated script and data into a single file, that can be posted on an FTP site or loaded to a CD-ROM, floppy disk, etc. See "Deploying Template" in the *Oracle8i Replication* manual for more information.

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 view.

The user that executes this public procedure will become the "registered" user of the instantiated template at the specified site.

Note: This procedure is used in performing an offline instantiation of a deployment template.

This procedure 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 snapshot). See these respective packages for more information on their usage.

Syntax

The parameter for the INSTANTIATE_OFFLINE function is described in Table 8–217, and the exception is described in Table 8–218. The syntax for this function is shown below:

```
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')
     return NUMBER
```

Table 8–217 Parameter for INSTANTIATE_OFFLINE

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Table 8–218 Exception for INSTANTIATE_OFFLINE

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_user	The name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, than the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and / or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e. type mismatch).

Returns

Table 8–219 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<system generated number>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE procedure

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots).

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, 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 view.

The user that executes this public procedure will become the "registered" user of the instantiated template at the specified site.

Syntax

The parameter for the INSTANTIATE_ONLINE function is described in Table 8–220, and the exception is described in Table 8–221. The syntax for this function is shown below:

```
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')
     return NUMBER
```

Table 8–220 Parameter for INSTANTIATE_ONLINE

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Table 8–221 Exception for INSTANTIATE_ONLINE

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_user	The name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, than the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and / or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e. type mismatch).

Returns

Table 8–222 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<system generated number>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

DBMS_REPCAT_RGT Package

Summary of Subprograms

Table 8–223 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
ALTER_REFRESH_TEMPLATE procedure on page 8-185	Allows the DBA to alter existing deployment templates.
ALTER_TEMPLATE_OBJECT procedure on page 8-187	Alters objects that have been added to a specified deployment template.
ALTER_TEMPLATE_PARM procedure on page 8-190	Allows the DBA to alter the parameters for a specific deployment template.
ALTER_USER_AUTHORIZATION procedure on page 8-192	Alters the contents of the DBA_REPCAT_TEMPLATE_AUTH view.
ALTER_USER_PARM_VALUE procedure on page 8-194	Changes existing parameter values that have been defined for a specific user.
COMPARE_TEMPLATES function on page 8-196	Allows a DBA to compare the contents of two deployment templates.
COPY_TEMPLATE function on page 8-198	Allows the DBA to copy a deployment template.
CREATE_OBJECT_FROM_EXISTING function on page 8-200	Creates a template object definition from existing database objects and adds it to a target deployment template.
CREATE_REFRESH_TEMPLATE function on page 8-203	Creates the deployment template, which allows you to define the template name, private / public status, and target refresh group.
CREATE_TEMPLATE_OBJECT function on page 8-206	Adds object definitions to a target deployment template container.
CREATE_TEMPLATE_PARM function on page 8-209	Creates parameters for a specific deployment template to allow custom data sets to be created at the remote snapshot site.
CREATE_USER_AUTHORIZATION function on page 8-212	Authorizes specific users to instantiate private deployment templates.
CREATE_USER_PARM_VALUE function on page 8-214	Pre-defines deployment template parameter values for specific users.

Table 8–223 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
DELETE_RUNTIME_PARMS procedure on page 8-217	Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.
DROP_ALL_OBJECTS procedure on page 8-218	Allows the DBA to drop all objects or specific object types from a deployment template.
DROP_ALL_TEMPLATE_PARMS procedure on page 8-219	Allows the DBA to drop template parameters for a specified deployment template.
DROP_ALL_TEMPLATE_SITES procedure on page 8-220	Removes all entries from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_ALL_TEMPLATES procedure on page 8-221	Removes all deployment templates at the site where the procedure is called.
DROP_ALL_USER_AUTHORIZATIONS procedure on page 8-222	Allows the DBA to drop all user authorizations for a specified deployment template.
DROP_ALL_USER_PARM_VALUES procedure on page 8-223	Drops user parameter values for a specific deployment template.
DROP_REFRESH_TEMPLATE procedure on page 8-225	Drops a deployment template.
DROP_SITE_INSTANTIATION procedure on page 8-226	Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_TEMPLATE_OBJECT procedure on page 8-227	Removes a template object from a specific deployment template.
DROP_TEMPLATE_PARM procedure on page 8-229	Removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view.
DROP_USER_AUTHORIZATION procedure on page 8-230	Removes a user authorization entry from the DBA_REPCAT_TEMPLATE_AUTH view.
DROP_USER_PARM_VALUE procedure on page 8-231	Removes a pre-defined user parameter value for a specific deployment template.
GET_RUNTIME_PARM_ID function on page 8-232	Retrieves an ID to be used when defining a runtime parameter value.
INSERT_RUNTIME_PARMS procedure on page 8-233	Defines runtime parameter values prior to instantiating a template.

Table 8–223 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
INSTANTIATE_OFFLINE function on page 8-235	Generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.
INSTANTIATE_ONLINE function on page 8-238	Generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online.
LOCK_TEMPLATE_EXCLUSIVE procedure on page 241	Prevents users from reading or instantiating the template when a deployment template is being updated or modified.
LOCK_TEMPLATE_SHARED procedure on page 8-242	Makes a specified deployment template read-only.

ALTER_REFRESH_TEMPLATE procedure

This procedure allows the DBA to alter existing deployment templates. Alterations may include defining a new deployment template name, a new refresh group, or a new owner and changing the public/private status.

Syntax

```
DBMS_REPCAT_RGT.ALTER_REFRESH_TEMPLATE (
   refresh_template_name      IN   VARCHAR2,
   new_owner                  IN   VARCHAR2 := '-',
   new_refresh_group_name     IN   VARCHAR2 := '-',
   new_refresh_template_name  IN   VARCHAR2 := '-',
   new_template_comment       IN   VARCHAR2 := '-',
   new_public_template        IN   VARCHAR2 := '-',
   new_last_modified          IN   DATE := to_date('1', 'J'),
   new_modified_by            IN   NUMBER := -1e-130);
```

Parameters

Table 8–224 ALTER_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template that you want to alter.
new_owner	The name of the new deployment template owner. Do not specify a value to keep the current owner.
new_refresh_group_name	If necessary, use this parameter to specify a new refresh group name that the template objects will be added to. Do not specify a value to keep the current refresh group.
new_refresh_template_name	Use this parameter to specify a new deployment template name. Do not specify a value to keep the current deployment template name.
new_template_comment	New deployment template comments. Do not specify a value to keep the current template comment.
new_public_template	Determines whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private). Do not specify a value to keep the current value.
new_last_modified	Contains the date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
new_modified_by	Contains the name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 8–225 ALTER_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.
dupl_refresh_template	A template with the specified name already exists. See the DBA_REPCAT_REFRESH_TEMPLATES view.

ALTER_TEMPLATE_OBJECT procedure

This procedure alters objects that have been added to a specified deployment template. The most common changes may include altering the object DDL and/or assigning the object to a different deployment template.

Changes made to the template is reflected only at new sites instantiating the deployment template. Remote sites that have already instantiated the template need to re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT (
   refresh_template_name       IN   VARCHAR2,
   object_name                 IN   VARCHAR2,
   object_type                 IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_object_name             IN   VARCHAR2 := '-',
   new_object_type             IN   VARCHAR2 := '-',
   new_ddl_text                IN   CLOB  := '-',
   new_master_rollback_seg     IN   VARCHAR2 := '-',
   new_flavor_id               IN   NUMBER := -1e-130);
```

Parameters

Table 8–226 ALTER_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the object that you want to alter.
object_name	Name of the template object that you want to alter.
object_type	Type of object that you want to alter.
new_refresh_template_name	Name of the new deployment template that you want to re-assign this object to. Do not specify a value to keep the object assigned to the current deployment template.
new_object_name	New name of the template object. Do not specify a value to keep the current object name.
new_object_type	If specified, then the new object type. Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER SEQUENCE DATABASE LINK
new_ddl_text	New object DDL for specified object. Do not specify any new DDL text to keep the current object DDL.
new_master_rollback_seg	New master rollback segment for specified object. Do not specify a value to keep the current rollback segment.
new_flavor_id	New flavor ID for the specified object. Do not specify a value to keep the current flavor ID.

Exceptions

Table 8–227 ALTER_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_flavor_id	Flavor ID specified is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 8–226 for a list of valid object types.
miss_template_object	Template object name specified is invalid or does not exist.
dupl_template_object	New template name specified in the new_refresh_template_name parameter already exists.

Usage Notes

Because the ALTER_TEMPLATE_OBJECT procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_TEMPLATE_OBJECT procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_TEMPLATE_OBJECT procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE SNAPSHOT snap_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 => 'SNAP_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 may include renaming the parameter or redefining the default value and prompt string.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
   refresh_template_name       IN   VARCHAR2,
   parameter_name              IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_parameter_name          IN   VARCHAR2 := '-',
   new_default_parm_value      IN   CLOB := NULL,
   new_prompt_string           IN   VARCHAR2 := '-',
   new_user_override           IN   VARCHAR2 := '-');
```

Parameters

Table 8–228 ALTER_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter that you want to alter.
parameter_name	Name of the parameter that you want to alter.
new_refresh_template_name	Name of the deployment template that the specified parameter should be re-assigned to (useful when you want to move a parameter from one template to another). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	New name of the template parameter. Do not specify a value to keep the current parameter name.
new_default_parm_value	New default value for the specified parameter. Do not specify a value to keep the current default value.
new_prompt_string	New prompt text for the specified parameter. Do not specify a value to keep the current prompt string.
new_user_override	Determines if the user can override the default value if prompted during the instantiation process (the user is prompted if no user parameter value has been defined for this parameter). Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to not allow an override.

Exceptions

Table 8–229 ALTER_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
dupl_template_parm	Combination of new_refresh_template_name and new_ parameter_name already exists.

Usage Notes

Because the ALTER_TEMPLATE_PARM procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_TEMPLATE_PARM procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_ TEMPLATE_PARM procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      new_default_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

ALTER_USER_AUTHORIZATION procedure

This procedure alters the contents of the DBA_REPCAT_TEMPLATE_AUTH view. Specifically, you can change user / deployment template authorization assignments. This procedure is helpful, for example, if an employee moves positions and requires the snapshot 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 8–230 ALTER_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
user_name	Name of the user whose authorization you want to alter.
refresh_template_name	Name of the deployment template that is currently assigned to the specified user that you want to alter.
new_user_name	Use this parameter to define a new user for this template authorization. Do not specify a value to keep the current user
new_refresh_template_name	The deployment template that the specified user (either the existing or, if specified, the new user) is authorized to instantiate. Do not specify a value to keep the current deployment template.

Exceptions

Table 8–231 ALTER_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user_ authorization	The combination of user_name and refresh_template_name values specified does not exist in the DBA_REPCAT_TEMPLATE_ AUTH view.
miss_user	The user name specified for the new_user_name or user_name parameter is invalid or does not exist.
miss_refresh_ template	The deployment template specified for the new_refresh_ template parameter is invalid or does not exist.
dupl_user_ authorization	A row already exists for the specified user name and deployment template name. See the DBA_REPCAT_AUTH_TEMPLATES view.

ALTER_USER_PARM_VALUE procedure

This procedure changes existing parameter values that have been defined for a specific user. This procedure is especially helpful if your snapshot environment uses assignment tables; simply change a user parameter value to quickly and securely change the data set of a remote snapshot site.

See "Deployment Template Design" in the *Oracle8i Replication* manual for more information on using assignment tables.

Syntax

```
DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
   refresh_template_name         IN   VARCHAR2,
   parameter_name                IN   VARCHAR2,
   user_name                     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2 := '-',
   new_parameter_name            IN   VARCHAR2 := '-',
   new_user_name                 IN   VARCHAR2 := '-',
   new_parm_value                IN   CLOB := NULL);
```

Parameters

Table 8–232 ALTER_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user parameter value that you want to alter.
parameter_name	Name of the parameter that you want to alter.
user_name	Name of the user whose parameter value you want to alter.
new_refresh_template_name	Name of the deployment template that the specified user parameter value should be re-assigned to (useful when you are authorizing a user for a different template). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	The new template parameter name. Do not specify a value to keep the user value defined for the existing parameter.
new_user_name	The new user name that this parameter value is for. Do not specify a value to keep the parameter value assigned to the current user.
new_parm_value	The new parameter value for the specified user parameter. Do not specify a value to keep the current parameter value.

Exceptions

Table 8–233 ALTER_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
miss_user	User name specified for the user_name or new_user_name parameters is invalid or does not exist.
miss_user_parm_ values	User parameter value specified does not exist.
dupl_user_parm_ values	New user parameter specified already exists.

Usage Notes

Because the ALTER_USER_PARM_VALUE procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_USER_PARM_VALUE procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_USER_PARM_VALUE procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      user_name => 'BOB',
      new_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

COMPARE_TEMPLATES function

This function allows a DBA to compare the contents of two deployment templates. Any discrepancies between the two deployment templates is stored in the USER_REPCAT_TEMP_OUTPUT table.

The COMPARE_TEMPLATES function returns a number that you specify in the WHERE clause when querying the USER_REPCAT_TEMP_OUTPUT table. For example, if the COMPARE_TEMPLATES procedure returns the number 10, you would execute the following SELECT statement to view all discrepancies between two specified templates (your SELECT statement returns no rows if the templates are identical):

```
SELECT text FROM user_repcat_temp_output
   WHERE output_id =  10 ORDER BY LINE;
```

The contents of the USER_REPCAT_TEMP_OUTPUT are lost after you disconnect or a ROLLBACK has been performed.

Syntax

```
DBMS_REPCAT_RGT.COMPARE_TEMPLATES (
   source_template_name    IN   VARCHAR2,
   compare_template_name   IN   VARCHAR2)
  return NUMBER;
```

Parameters

Table 8–234 COMPARE_TEMPLATES Function Parameters

Parameter	Description
source_template_name	Name of the first deployment template to be compared.
compare_template_name	Name of the second deployment template to be compared.

Exceptions

Table 8–235 COMPARE_TEMPLATES Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name to be compared is invalid or does not exist.

Returns

Table 8–236 COMPARE_TEMPLATES Function Returns

Return Value	Description
<*system generated number*>	Specify the number returned for the output_id value when you select from the USER_REPCAT_TEMP_OUTPUT view to view the discrepancies between the compared templates.

COPY_TEMPLATE function

This function allows the DBA to copy a deployment template. COPY_TEMPLATE is helpful when a new deployment template will use many of the object contained in an existing deployment template. This function copies the deployment template, template objects, template parameters, and user parameter values. The DBA can optionally have the function copy the user authorizations for this template. The number returned by this function is used internally by Oracle to manage deployment templates.

Note: The values in the DBA_REPCAT_TEMPLATE_SITES view are not copied.

This function also allows the DBA to copy a deployment template to another master site, which is helpful for deployment template distribution and to split network loads between multiple sites.

Syntax

```
DBMS_REPCAT_RGT.COPY_TEMPLATE (
   old_refresh_template_name     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2,
   copy_user_authorizations      IN   VARCHAR2,
   dblink                        IN   VARCHAR2 := NULL)
   return NUMBER;
```

Parameters

Table 8–237 COPY_TEMPLATE Function Parameters

Parameter	Description
old_refresh_ template_name	Name of the deployment template to be copied.
new_refresh_ template_name	Name of the new deployment template.
copy_user_ authorizations	Specifies whether the template authorizations for the original template should be copied for the new deployment template. Valid values for this parameter are 'Y', 'N' and NULL. **Note**: All users must exist at the target database.
dblink	Optionally defines where the deployment template should be copied from (this is helpful to distribute deployment templates to other master sites). If none is specified, then the deployment template is copied from the local master site.

Exceptions

Table 8–238 COPY_TEMPLATE Function Exceptions

Exception	Description
miss_refresh_ template	Deployment template name to be copied is invalid or does not exist.
dupl_refresh_ template	Name of the new refresh template specified already exists.
bad_copy_auth	Value specified for the copy_user_authorization parameter is invalid. Valid values are 'Y', 'N', and NULL.

Returns

Table 8–239 COPY_TEMPLATES Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_OBJECT_FROM_EXISTING function

This function creates a template object definition from existing database objects and adds it to a target deployment template. The object DDL that created the original database object is executed when the target deployment template is instantiated at the remote snapshot site. This is ideal for adding existing triggers and procedures to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_OBJECT_FROM_EXISTING(
     refresh_template_name  IN   VARCHAR2,
     object_name            IN   VARCHAR2,
     sname                  IN   VARCHAR2,
     oname                  IN   VARCHAR2,
     otype                  IN   VARCHAR2)
     return NUMBER
```

Parameters

Table 8–240 CREATE_OBJECT_FROM_EXISTING Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to add this object to.
object_name	If necessary, the new name of the existing object that you are adding to your deployment template (allows you to define a new name for an existing object).
sname	The schema that contains the object that you are creating your template object from.
oname	Name of the object that you are creating your template object from.
otype	The type of database object that you are adding to the template (i.e., PROCEDURE, TRIGGER, etc.). The object type must be specified using the following numerical identifiers (DATABASE LINK or SNAPSHOT are not a valid object types for this function): SEQUENCE PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER

Exceptions

Table 8–241 CREATE_OBJECT_FROM_EXISTING Function Exceptions

Exception	Description
miss_refresh_template	The specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of existing deployment templates.
bad_object_type	The object type is specified incorrectly (see Table 8–246 for more information).
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.
objectmissing	Existing object specified does not exist.

Returns

Table 8–242 CREATE_OBJECT_FROM_EXISTING Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_REFRESH_TEMPLATE function

This function creates the deployment template, which allows you to define the template name, private / public status, and target refresh group. Each time that you create a template object, user authorization, or template parameter, you reference the deployment template created with this function. This function adds a row to the DBA_REPCAT_REFRESH_TEMPLATES view. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
   owner                 IN   VARCHAR2,
   refresh_group_name    IN   VARCHAR2,
   refresh_template_name IN   VARCHAR2,
   template_comment      IN   VARCHAR2 := NULL,
   public_template       IN   VARCHAR2 := NULL,
   last_modified         IN   DATE := SYSDATE,
   modified_by           IN   VARCHAR2 := USER,
   creation_date         IN   DATE := SYSDATE,
   created_by            IN   VARCHAR2 := USER)
  return NUMBER;
```

Parameters

Table 8–243 CREATE_REFRESH_TEMPLATE Function Parameters

Parameter	Description
owner	User name of the deployment template owner is specified with this parameter. If an owner is not specified, then the name of the user creating the template is automatically used.
refresh_group_name	Name of the refresh group that is created when this template is instantiated. All objects created by this template are assigned to the specified refresh group.
refresh_template_name	Name of the deployment template that you are creating. This name is referenced in all activities that involve this deployment template.
template_comment	User comments defined with this are listed in the DBA_REPCAT_REFRESH_TEMPLATES view.
public_template	Specifies whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private).
last_modified	The date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
modified_by	Name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.
creation_date	The date that this deployment template was created. If a value is not specified, then the current date is automatically used.
created_by	Name of the user who created this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 8–244 CREATE_REFRESH_TEMPLATE Function Exceptions

Exception	Description
dupl_refresh_template	A template with the specified name already exists. See the DBA_REPCAT_REFRESH_TEMPLATES view to see a list of existing templates.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.

Returns

Table 8–245 CREATE_REFRESH_TEMPLATE Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_TEMPLATE_OBJECT function

This function adds object definitions to a target deployment template container. The specified object DDL is executed when the target deployment template is instantiated at the remote snapshot site. In addition to adding snapshots, this function can add tables, procedures, and other objects to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2,
   ddl_text               IN   CLOB,
   master_rollback_seg    IN   VARCHAR2 := NULL,
   flavor_id              IN   NUMBER := -1e-130)
  return NUMBER;
```

Parameters

Table 8–246 CREATE_TEMPLATE_OBJECT Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to add this object to.
object_name	Name of the template object that you are creating.
object_type	The type of database object that you are adding to the template (i.e., SNAPSHOT, TRIGGER, PROCEDURE, etc.). Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM MATERIALIZED VIEW SEQUENCE DATABASE LINK TRIGGER
ddl_text	Contains the DDL that creates the object that you are adding to the template. Be sure to end your DDL with a semi-colon. (Remember, you can use a colon (:) to create a template parameter for your template object; see "Creating Snapshots with Deployment Templates" in the *Oracle8i Replication* book for more information.
master_rollback_seg	Specifies the name of the rollback segment to use when executing the defined object DDL at the remote snapshot site.
flavor_id	Defines the flavor ID for this template object.

Exceptions

Table 8–247 CREATE_TEMPLATE_OBJECT Function Exceptions

Exception	Description
miss_refresh_template	Specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of existing deployment templates.
bad_object_type	Object type is specified incorrectly. See Table 8–246 for a list of valid object types.
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.

Returns

Table 8–248 CREATE_TEMPLATE_OBJECT Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because CREATE_TEMPLATE_OBJECT utilizes a CLOB, you need to utilize the DBMS_LOB package when using the CREATE_TEMPLATE_OBJECT function. The following example illustrates how to use the DBMS_LOB package with the CREATE_TEMPLATE_OBJECT function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE SNAPSHOT snap_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 => 'snap_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 snapshot 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 on page 8-190 for more information). The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
   parameter_name         IN   VARCHAR2,
   default_parm_value     IN   CLOB := NULL,
   prompt_string          IN   VARCHAR2 := NULL,
   user_override          IN   VARCHAR2 := NULL)
   return NUMBER;
```

Parameters

Table 8–249 CREATE_TEMPLATE_PARM Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to create the parameter for.
parameter_name	Name of the parameter you are creating.
default_parm_value	Default values for this parameter are defined using this parameter. If a user parameter value or runtime parameter value is not present, then this default value is used during the instantiation process.
prompt_string	The descriptive prompt text that is displayed for this template parameter during the instantiation process.
user_override	Determines if the user can override the default value if prompted during the instantiation process (the user is prompted if no user parameter value has been defined for this parameter). Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to not allow an override.

Exceptions

Table 8–250 CREATE_TEMPLATE_PARM Function Exceptions

Exception	Description
miss_refresh_template	The specified refresh template name is invalid or missing.
dupl_template_parm	A parameter of the same name has already been defined for the specified deployment template.

Returns

Table 8–251 CREATE_TEMPLATE_PARM Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because the CREATE_TEMPLATE_PARM function utilizes a CLOB, you need to utilize 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_AUTH_TEMPLATES 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 8–252 CREATE_USER_AUTHORIZATION Function Parameters

Parameter	Description
user_name	Name of the user that you want to authorize to instantiate the specified template. Specify multiple users by separating user names with a comma (i.e., 'john, mike, bob')
refresh_template_name	Name of the template that you want to authorize the specified user to instantiate.

Exceptions

Table 8–253 CREATE_USER_AUTHORIZATION Function Exceptions

Exception	Description
miss_user	User name supplied is invalid or does not exist.
miss_refresh_template	Refresh template name supplied is invalid or does not exist.
dupl_user_authorization	An authorization has already been created for the specified user and deployment template. See the DBA_REPCAT_AUTH_TEMPLATES view for a listing of template authorizations.

Returns

Table 8–254 CREATE_USER_AUTHORIZATION Function Returns

Return Value	Description
<*system generated number*>	System generated number is used internally by Oracle.

CREATE_USER_PARM_VALUE function

This function is used to pre-define deployment template parameter values for specific users. For example, if you want to pre-defined the region parameter as WEST for user 33456, then you would use the this function.

Any values specified with this function take precedence over default values specified for the template parameter. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
   refresh_template_name    IN   VARCHAR2,
   parameter_name           IN   VARCHAR2,
   user_name                IN   VARCHAR2,
   parm_value               IN   CLOB := NULL)
  return NUMBER;
```

Parameters

Table 8–255 CREATE_USER_PARM_VALUE Function Parameters

Parameter	Description
refresh_template_name	Specifies the name of the deployment template that contains the parameter you are creating a user value for.
parameter_name	Name of the template parameter that you are defining a user parameter value for.
user_name	Specifies the name of the user that you are pre-defining a parameter value for.
parm_value	The pre-defined parameter value that will be used during the instantiation process initiated by the specified user.

Exceptions

Table 8–256 CREATE_USER_PARM_VALUE Function Exceptions

Exception	Description
miss_refresh_ template	Specified deployment template name is invalid or missing.
dupl_user_parm_ values	A parameter value for the specified user, parameter, and deployment template has already been defined. Query the DBA_ REPCAT_USER_PARMS view for a listing of existing user parameter values.
miss_template_parm	Specified deployment template parameter name is invalid or missing.
miss_user	Specified user name is invalid or missing.

Returns

Table 8–257 CREATE_USER_PARM_VALUE Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because the CREATE_USER_PARM_VALUE function utilizes a CLOB, you need to utilize the DBMS_LOB package when using the this function. The following example illustrates how to use the DBMS_LOB package with the CREATE_USER_PARM_VALUE function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE(
        refresh_template_name => 'rgt_personnel',
        parameter_name => 'region',
        user_name => 'BOB',
        user_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

DELETE_RUNTIME_PARMS procedure

Use this procedure before instantiating a deployment template to delete a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Syntax

```
DBMS_REPCAT_RGT.DELETE_RUNTIME_PARMS(
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2);
```

Parameters

Table 8–258 DELETE_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	Specifies the ID that you previously assigned the runtime parameter value to (this value was retrieved using the GET_RUNTIME_PARM_ID function).
parameter_name	Specifies the name of the parameter value that you want to drop (query the DBA_REPCAT_TEMPLATE_PARMS for a list of deployment template parameters).

Exceptions

Table 8–259 DELETE_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_template_parm	The specified deployment template parameter name is invalid or missing.

DROP_ALL_OBJECTS procedure

This procedure allows the DBA to drop all objects or specific object types from a deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_OBJECTS (
   refresh_template_name   IN   VARCHAR2,
   object_type             IN   VARCHAR2 := NULL);
```

Parameters

Table 8–260 DROP_ALL_OBJECTS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.
object_type	If NULL, then all objects in the template are dropped. If an object type is specified, then only objects of that type are dropped. Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM MATERIALIZED VIEW SEQUENCE DATABASE LINK TRIGGER

Exceptions

Table 8–261 DROP_ALL_OBJECTS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 8–260 for a list of valid object types.

DROP_ALL_TEMPLATE_PARMS procedure

This procedure allows the DBA to drop template parameters for a specified deployment template. The DBA can use this procedure to drop all parameters that are not referenced by a template objects or drop all objects that reference a parameter and 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 8–262 DROP_ALL_TEMPLATE_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameters that you want to drop.
drop_objects	If no value is specified, then this defaults to N, which drops all parameters not referenced by a template object. If Y is specified, then all objects that reference a template parameter and the template parameters themselves are dropped.

Exceptions

Table 8–263 DROP_ALL_TEMPLATE_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATE_SITES procedure

This procedure removes all entries from the DBA_REPCAT_TEMPLATE_SITES view, which keeps a record of sites that have instantiated a particular deployment template.

Caution: This is a dangerous procedure that cannot be undone. Additionally, Oracle Lite sites that have instantiated the dropped template will no longer be able to refresh their snapshots.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_SITES (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 8–264 DROP_ALL_TEMPLATE_SITES Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the sites that you want to drop.

Exceptions

Table 8–265 DROP_ALL_TEMPLATE_SITES Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATES procedure

This procedure removes all deployment templates at the site where the procedure is called.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATES;
```

Parameters

None

DROP_ALL_USER_AUTHORIZATIONS procedure

This procedure allows the DBA to drop all user authorizations for a specified deployment template. Executing this procedure removes rows from the DBA_REPCAT_AUTH_TEMPLATES view.

This procedure might be implemented after converting a private template to a public template and the user authorizations are no longer required.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_AUTHORIZATIONS (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 8–266 DROP_ALL_USER_AUTHORIZATIONS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.

Exceptions

Table 8–267 DROP_ALL_USER_AUTHORIZATIONS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_USER_PARM_VALUES procedure

This procedure drops user parameter values for a specific deployment template. This procedure is very flexible in allowing the DBA to define a set of user parameter values to be deleted. For example, defining the following parameters have the effect:

`refresh_template_name`: drops all user parameters for the specified deployment template.

`refresh_template_name`, `user_name`: drops all of the specified user parameters for the specified deployment template.

`refresh_template_name`, `parameter_name`: drops all user parameter values for the specified deployment template parameter.

`refresh_template_name`, `parameter_name`, `user_name`: drops the specified user's value for the specified deployment template parameter (equivalent to DROP_USER_PARM).

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_PARMS (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2,
   parameter_name          IN   VARCHAR2);
```

Parameters

Table 8–268 DROP_ALL_USER_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter values that you want to drop.
user_name	Name of the user whose parameter values you want to drop.
parameter_name	Template parameter that contains the values that you want to drop.

Exceptions

Table 8–269 DROP_ALL_USER_PARMS Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_ values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

DROP_REFRESH_TEMPLATE procedure

This procedure drops a deployment template. Dropping a deployment template has a cascading effect, removing all related template parameters, user authorizations, template objects, and user parameters (this procedure does not drop template sites).

Syntax

```
DBMS_REPCAT_RGT.DROP_REFRESH_TEMPLATE (
   refresh_template_name  IN   VARCHAR2);
```

Parameters

Table 8–270 DROP_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be dropped.

Exceptions

Table 8–271 DROP_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of deployment templates.

DROP_SITE_INSTANTIATION procedure

Purpose

This procedure drops a template instantiation at a target site. This procedure removes all related meta data at the master site and disables the specified site from refreshing their snapshots.

Syntax

The parameter for the DROP_SITE_INSTANTIATION procedure is described in Table 8–270, and the exception is described in Table 8–271. The syntax for this procedure is shown below:

```
DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
     refresh_template_name  IN   VARCHAR2,
     user_name              IN   VARCHAR2,
     site_name              IN   VARCHAR2,
     repapi_site_id         IN   NUMBER := -1e-130)
```

Table 8–272 Parameter for DROP_SITE_INSTANTIATION

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
user_name	Enter the name of the user that originally instantiated the template at the remote snapshot site. Query the REPCAT_TEMPLATE_SITES view to see the users that instantiated templates (see the "REPCAT_TEMPLATE_SITES View" section on page 9-11 for more information).
site_name	Identifies the Oracle server site where you want to drop the specified template instantiation (if you specify a SITE_NAME, do not specify a REPAPI_SITE_ID).
repapi_site_id	Identifies the REPAPI location where you want to drop the specified template instantiation (if you specify a REPAPI_SITE_ID, do not specify a SITE_NAME).

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 snapshot 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 need to re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2);
```

Parameters

Table 8–273 DROP_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you are dropping the object from.
object_name	Name of the template object to be dropped.
object_type	The type of object that is to be dropped. Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM MATERIALIZED VIEW SEQUENCE DATABASE LINK TRIGGER

Exceptions

Table 8–274 DROP_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_template_object	The template object specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_OBJECT view to see a list of deployment template objects.

DROP_TEMPLATE_PARM procedure

This procedure removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view. This procedure is helpful when you have dropped a template object and a particular parameter is no longer needed.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
   parameter_name         IN   VARCHAR2);
```

Parameters

Table 8–275 DROP_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	The deployment template name that has the parameter that you want to drop
parameter_name	Name of the parameter that you want to drop.

Exceptions

Table 8–276 DROP_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_template_parm	The parameter name specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_PARMS view to see a list of template parameters.

DROP_USER_AUTHORIZATION procedure

This procedure removes a user authorization entry from the DBA_REPCAT_TEMPLATE_AUTH 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 8-222 for additional information.

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2);
```

Parameters

Table 8–277 DROP_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that the user's authorization is being removed from.
user_name	Name of the user whose authorization is being removed.

Exceptions

Table 8–278 DROP_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user	Specified user name is invalid or does not exist.
miss_user_authorization	Specified user and deployment template combination does not exist. Query the DBA_REPCAT_TEMPLATE_AUTH view to see a list of user / deployment template authorizations.
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_USER_PARM_VALUE procedure

This procedure removes a pre-defined 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 8–279 DROP_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the parameter value that you want to drop.
parameter_name	Parameter name that contains the pre-defined value that you want to drop.
user_name	Name of the user whose parameter value you want to drop.

Exceptions

Table 8–280 DROP_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

GET_RUNTIME_PARM_ID function

This function retrieves an ID to be used when defining a runtime parameter value. All runtime parameter values are assigned to this ID and are also used during the instantiation process.

Syntax

```
DBMS_REPCAT_RGT.GET_RUNTIME_PARM_ID
  RETURN NUMBER;
```

Parameters

None

Returns

Table 8–281 GET_RUNTIME_PARM_ID Function Returns

Return Value	Corresponding Datatype
<system generated number>	Runtime parameter values are assigned to the system generated number and is also used during the instantiation process.

INSERT_RUNTIME_PARMS procedure

This procedure defines runtime parameter values prior to instantiating a template. This procedure should be used to define parameter values when no user parameter values have been defined and you do not want to accept the default parameter values.

Before using the this procedure, be sure to execute the GET_RUNTIME_PARM_ID function to retrieve a parameter ID to be used when inserting a runtime parameter. This ID is used for defining runtime parameter values and instantiating deployment template.

Syntax

```
DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS (
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2,
   parameter_value    IN   CLOB);
```

Parameters

Table 8–282 INSERT_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	The ID retrieved by the GET_RUNTIME_PARM_ID function. This ID is also used when instantiating the deployment template (be sure to use the same ID for all parameter values for a deployment template).
parameter_name	Name of the template parameter that you are defining a runtime parameter value for (query the DBA_REPCAT_TEMPLATE_PARMS view for a list of template parameters).
parameter_value	The runtime parameter value that you want to use during the deployment template instantiation process.

Exceptions

Table 8–283 INSERT_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.

Table 8–283 INSERT_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_user	The user name specified is invalid or does not exist.
miss_user_parm_values	The deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

Usage Notes

Because the this procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the INSERT_RUNTIME_PARMS procedure. The following example illustrates how to use the DBMS_LOB package with the INSERT_RUNTIME_PARMS procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS(
      runtime_parm_id => 20,
      parameter_name => 'region',
      parameter_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

INSTANTIATE_OFFLINE function

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots). This procedure needs to 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 view.

Note: This procedure is used in performing an offline instantiation of a deployment template. Additionally, this procedure is for replication administrators that are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the "INSTANTIATE_OFFLINE procedure" function, described on page 8-178.

This procedure 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 snapshot). See these respective packages for more information on their usage.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
     refresh_template_name   IN   VARCHAR2,
     site_name               IN   VARCHAR2,
     user_name               IN   VARCHAR2 := NULL,
     runtime_parm_id         IN   NUMBER := -1e-130,
     next_date               IN   DATE := SYSDATE,
     interval                IN   VARCHAR2 :'SYSDATE + 1')
     return NUMBER
```

Parameters

Table 8–284 INSTANTIATE_OFFLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 8–285 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and/or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 8–286 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<*system generated number*>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE function

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots). This procedure needs to 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 view.

Note: This procedure is for replication administrators that are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the "INSTANTIATE_ONLINE procedure" function, described on page 8-180.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
   refresh_template_name   IN   VARCHAR2,
   site_name               IN   VARCHAR2 := NULL,
   user_name               IN   VARCHAR2 := NULL,
   runtime_parm_id         IN   NUMBER := -1e-130,
   next_date               IN   DATE := SYSDATE,
   interval                IN   VARCHAR2 :'SYSDATe + 1')
  return NUMBER;
```

Parameters

Table 8–287 INSTANTIATE_ONLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 8–288 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and/or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 8–289 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<system generated number>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

LOCK_TEMPLATE_EXCLUSIVE procedure

When a deployment template is being updated or modified, you should use the LOCK_TEMPLATE_EXCLUSIVE procedure to prevent users from reading or instantiating the template.

The lock will be released when a ROLLBACK or COMMIT is performed.

Note: This procedure should be executed before you make any modifications to your deployment template.

Syntax

The syntax for this procedure is shown below:

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_EXCLUSIVE()
```

LOCK_TEMPLATE_SHARED procedure

The LOCK_TEMPLATE_SHARED procedure is used to make a specified deployment template "read-only." This procedure should be called before instantiating a template, as this will ensure that nobody can change the deployment template while it is being instantiated.

The lock will be released when a ROLLBACK or COMMIT is performed.

Syntax

The syntax for this procedure is shown below:

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_SHARED()
```

DBMS_REPUTIL Package

Summary of Subprograms

Table 8–290 DBMS_REPUTIL Package Subprograms

Subprogram	Description
REPLICATION_OFF procedure on page 8-244	modifies tables without replicating the modifications to any other sites in the replicated environment, or disables row-level replication when using procedural replication.
REPLICATION_ON procedure on page 8-245	Re-enables replication of changes after replication has been temporarily suspended.
REPLICATION_IS_ON function on page 8-246	Determines whether or not replication is running.
FROM_REMOTE function on page 8-247	Returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures.
GLOBAL_NAME function on page 8-248	Determines the global database name of the local database (the global name is the returned value).
MAKE_INTERNAL_PKG procedure on page 8-249	Synchronizes internal packages and tables in the replication catalog. This procedure is executed under the direction of Oracle Worldwide Support only.
SYNC_UP_REP procedure on page 8-250	Synchronizes internal triggers and tables/snapshots in the replication catalog. This procedure is executed under the direction of Oracle Worldwide Support only.

REPLICATION_OFF procedure

This procedure modifies tables without replicating the modifications to any other sites in the replicated environment, or disables row-level replication when using procedural replication. In general, you should suspend replication activity for all master groups in your replicated 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. FALSE indicates that replication is disabled at the current site for the replicated master group.

The returning value of this function is set by calling the REPLICATION_ON or REPLICATION_OFF procedures in the DBMS_REPUTIL package.

Syntax

```
DBMS_REPUTIL.REPLICATION_IS_ON
  return BOOLEAN;
```

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

```
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

```
DBMS_REPUTIL.GLOBAL_NAME
   return VARCHAR2;
```

Parameters

None

MAKE_INTERNAL_PKG procedure

This procedure synchronizes the existence of an internal package with a table in the replication catalog. If the table has replication support, execute this procedure to create the internal package. If replication support does not exist, destroy any related internal package.

Warning: This procedure should only be executed under the guidance of Oracle Worldwide Support.

Syntax

```
DBMS_REPUTIL.MAKE_INTERNAL_PKG (
   canon_sname    IN   VARCHAR2
   canon_oname    IN   VARCHAR2);
```

Parameters

Table 8–291 MAKE_INTERNAL_PKG Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table to be synchronized. This parameter value must be cannonically definined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table to be synchronized. This parameter value must be cannonically definined (capitalization must match object and must not be enclosed in double quotes).

SYNC_UP_REP procedure

This procedure synchronizes the existence of an internal trigger with a table or snapshot in the replication catalog. If the table or snapshot has replication support, execute this procedure to create the internal replication trigger. If replication support does not exist, destroy any related internal trigger.

Warning: This procedure should only be executed under the guidance of Oracle Worldwide Support.

Syntax

```
DBMS_REPUTIL.SYNC_UP_REP (
   canon_sname     IN   VARCHAR2
   canon_oname     IN   VARCHAR2);
```

Parameters

Table 8–292 SYNC_UP_REP Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table or snapshot to be synchronized. This parameter value must be cannonically definined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table or snapshot to be synchronized. This parameter value must be cannonically definined (capitalization must match object and must not be enclosed in double quotes).

DBMS_SNAPSHOT Package

Summary of Subprograms

Table 8–293 DBMS_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_TABLE_REORGANIZATION procedure on page 8-252	Performs a process to preserve snapshot data needed for refresh.
END_TABLE_REORGANIZATION on page 8-253	Ensures that the snapshot data for the master table is valid and that the master table is in the proper state.
I_AM_A_REFRESH function on page 8-254	Returns the value of the I_AM_REFRESH package state.
PURGE_DIRECT_LOAD_LOG procedure on page 8-255	Purges rows from the direct loader log after they are no longer needed by any snapshots (used with data warehousing).
PURGE_LOG procedure on page 8-256	Purges rows from the snapshot log.
PURGE_SNAPSHOT_FROM_LOG procedure on page 8-257	Purges rows from the snapshot log.
REFRESH procedure on page 8-259	Consistently refreshes one or more snapshots that are not members of the same refresh group.
REFRESH_ALL_MVIEWS procedure on page 8-261	Refreshes all snapshots that have not been refreshed because the most recent bulk load to a dependent detail table.
REFRESH_DEPENDENT procedure on page 8-263	Refreshes all table-based snapshots that depend on a specified detail table or list of detail tables.
REGISTER_SNAPSHOT procedure on page 8-265	Enables the administration of individual snapshots.
UNREGISTER_SNAPSHOT procedure on page 8-267	Enables the administration of individual snapshots. Invoked at a master site to unregister a snapshot.

BEGIN_TABLE_REORGANIZATION procedure

This procedure performs a process to preserve snapshot data needed for refresh. It must be called before a master table is reorganized.

Additional Information: See "Administering a Replicated Environment" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2
   tabname      IN   VARCHAR2);
```

Parameters

Table 8–294 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

END_TABLE_REORGANIZATION

This procedure must be called after a master table is reorganized. It ensures that the snapshot data for the master table is valid and that the master table is in the proper state.

Additional Information: See "Administering a Replicated Environment" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_SNAPSHOT.END_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2
   tabname      IN   VARCHAR2);
```

Parameters

Table 8–295 END_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

I_AM_A_REFRESH function

This function returns the value of the I_AM_REFRESH package state.

Syntax

```
DBMS_SNAPSHOT.I_AM_A_REFRESH
  RETURN BOOLEAN;
```

Parameters

None

Returns

A return value of TRUE indicates that all local replication triggers for snapshots 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.

PURGE_DIRECT_LOAD_LOG procedure

This procedure remove entries from the direct loader log after they are no longer needed for any known snapshot (materialized view). This procedure will usually be used in environments using Oracle's Data Warehousing technology. For more information, see *Oracle8i Tuning*.

Syntax

```
DBMS_SNAPSHOT.PURGE_DIRECT_LOAD_LOG ();
```

PURGE_LOG procedure

This procedure purges rows from the snapshot log.

Syntax

```
DBMS_SNAPSHOT.PURGE_LOG (
   master        IN   VARCHAR2,
   num           IN   BINARY_INTEGER := 1,
   flag          IN   VARCHAR2       := 'NOP');
```

Parameters

Table 8–296 PURGE_LOG Procedure Parameters

Parameter	Description
master	Name of the master table.

num	Number of least recently refreshed snapshots whose rows you want to remove from snapshot log. For example, the following statement deletes rows needed to refresh the two least recently refreshed snapshots:
	`dbms_snapshot.purge_log('master_table', 2);`
	To delete all rows in the snapshot log, indicate a high number of snapshots to disregard, as in this example:
	`dbms_snapshot.purge_log('master_table',9999);`
	This statement completely purges the snapshot log that corresponds to MASTER_TABLE if fewer than 9999 snapshots are based on MASTER_TABLE. A simple snapshot whose rows have been purged from the snapshot log must be completely refreshed the next time it is refreshed.

flag	Specify DELETE to guarantee that rows are deleted from the snapshot log for at least one snapshot. This argument can override the setting for the argument num. For example, the following statement deletes rows from the least recently refreshed snapshot that actually has dependent rows in the snapshot log: `dbms_snapshot.purge_log('master_table',1,'DELETE');`

PURGE_SNAPSHOT_FROM_LOG procedure

This procedure is called on the master site to delete the rows in snapshot refresh related data dictionary tables maintained at the master site for the specified snapshot identified by its `snapshot_id` or the combination of the `snapowner`, `snapname`, and the `snapsite`. If the snapshot specified is the oldest snapshot to have refreshed from any of the master tables, then the snapshot log is also purged. This procedure does not unregister the snapshot.

In case there is an error while purging one of the snapshot logs, the successful purge operations of the previous snapshot logs are not rolled back. This is to minimize the size of the snapshot logs. In case of an error, this procedure can be invoked again until all the snapshot logs are purged.

Syntax

```
DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   snapshot_id   IN   BINARY_INTEGER  |
   snapowner     IN   VARCHAR2,
   snapname      IN   VARCHAR2,
   snapsite      IN   VARCHAR2);
```

Parameters

Table 8–297 PURGE_SNAPSHOT_FROM_LOG Procedure Parameters

Parameter	Description
snapshot_id	If you want to execute this procedure based on the ID of the target snapshot, specify the snapshot ID using the `snapshot_id` parameter. Query the DBA_SNAPSHOT_LOGS view at the snapshot log site for a listing of snapshot IDs. Executing this procedure based on the snapshot ID is useful if the target snapshot is not listed in the list of registered snapshots (DBA_REGISTERED_SNAPSHOTS). If you specify a snapshot ID, do not specify values for the `snapowner`, `snapname`, or `snapsite` parameters.
snapowner	If do not specify a snapshot_id, enter the owner of the target snapshot using the `snapowner` parameter. Query the DBA_ REGISTERED_SNAPSHOTS view at the snapshot log site to view the snapshot owners.
snapname	If do not specify a snapshot_id, enter the name of the target snapshot using the `snapname` parameter. Query the DBA_ REGISTERED_SNAPSHOTS view at the snapshot log site to view the snapshot names.
snapsite	If do not specify a snapshot_id, enter the site of the target snapshot using the `snapsite` parameter. Query the DBA_REGISTERED_ SNAPSHOTS view at the snapshot log site to view the snapshot sites.

REFRESH procedure

This procedure refreshes a list of snapshots.

Syntax

```
DBMS_SNAPSHOT.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);
```

Parameters

Table 8–298 REFRESH Procedure Parameters

Parameter	Description
list tab	Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your local database. Alternatively, you may pass in a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a snapshot.
method	A string of refresh methods indicating how to refresh the listed snapshots. 'F' or 'f' indicates fast refresh, '?' indicates force refresh, 'C' or 'c' indicates complete refresh, and 'A' or 'a' indicates always refresh, If a snapshot does not have a corresponding refresh method (that is, if more snapshots are specified than refresh methods), then that snapshot is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `dbms_snapshot.refresh ('s_emp,s_dept,scott.s_salary','CF');` performs a complete refresh of the S_EMP snapshot, a fast refresh of the S_DEPT snapshot, and a default refresh of the SCOTT.S_SALARY snapshot.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.

Table 8–298 REFRESH Procedure Parameters

Parameter	Description
push_deferred_rpc	Used by updatable snapshots only. Set this parameter to TRUE if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	If this parameter is TRUE, an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure will continue to refresh other snapshots if it fails while refreshing a snapshot.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set purge to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, set purge to do not purge and occasionally execute PUSH with purge set to aggressive to reduce the queue.
parallelism	0 means serial propagation, n > 0 means parallel propagation with n parallel server processes, and 1 means parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set this parameter unless directed to do so by Oracle Worldwide Support.
atomic_refresh	If this parameter is set to TRUE, then the list of snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots will be updated. If this parameter is set to FALSE, then each of the snapshots will be refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE and the Summary Management option is not purchased, then an error is raised.

REFRESH_ALL_MVIEWS procedure

This procedure refreshes all snapshots (materialized views) with the following properties:

- the snapshot has not been refreshed since the most recent change to a detail table on which it depends
- the snapshot and all of the detail tables on which it depends are local
- the snapshot is in the view DBA_MVIEW_ANALYSIS

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_SNAPSHOT.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 8–299 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
number_of_failures	Returns the number of failures that occurred during processing.
method	A single refresh method indicating the type of refresh to perform for each snapshot 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. If no method is specified, a snapshot is refreshed according to its default refresh method.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.
refresh_after_errors	If this parameter is TRUE, an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure will continue to refresh other snapshots if it fails while refreshing a snapshot.

Table 8–299 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
atomic_refresh	If this parameter is set to TRUE, then the refreshed snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots will be updated. If this parameter is set to FALSE, then each of the refreshed snapshots will be refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE and the Summary Management option is not purchased, then an error is raised.

REFRESH_DEPENDENT procedure

This procedure refreshes all snapshots (materialized views) with the following properties:

- the snapshot depends on a detail table in the list of specified detail tables
- the snapshot has not been refreshed since the most recent change to a detail table on which it depends
- the snapshot and all of the detail tables on which it depends are local
- the snapshot is in the view DBA_MVIEW_ANALYSIS

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_SNAPSHOT.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);
```

Parameters

Table 8–300 REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
number_of_failures	Returns the number of failures that occurred during processing.
list tab	Comma-separated list of detail tables on which snapshots can depend. (Synonyms are not supported.) These tables and the snapshots that depend on them can be located in different schemas. However, all of the tables and snapshots must be in your local database. Alternatively, you may pass in a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table.

Table 8–300 REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
method	A string of refresh methods indicating how to refresh the dependent snapshots. All of the snapshots 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. If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any snapshot that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `dbms_snapshot.refresh_dependent ('emp,dept,scott.salary','CF');` performs a complete refresh of the snapshots that depend on the EMP table, a fast refresh of the snapshots that depend on the DEPT table, and a default refresh of the snapshots that depend on the SCOTT.SALARY table.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.
refresh_after_errors	If this parameter is TRUE, an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure will continue to refresh other snapshots if it fails while refreshing a snapshot.
atomic_refresh	If this parameter is set to TRUE, then the refreshed snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots will be updated. If this parameter is set to FALSE, then each of the refreshed snapshots will be refreshed in separate transactions. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE and the Summary Management option is not purchased, then an error is raised.

REGISTER_SNAPSHOT procedure

This procedure enables the administration of individual snapshots.

Syntax

```
DBMS_SNAPSHOT.REGISTER_SNAPSHOT (
   snapowner    IN   VARCHAR2,
   snapname     IN   VARCHAR2,
   snapsite     IN   VARCHAR2,
   snapshot_id IN   DATE | BINARY_INTEGER,
   flag         IN   BINARY_INTEGER,
   qry_txt      IN   VARCHAR2,
   rep_type     IN   BINARY_INTEGER := DBMS_SNAPSHOT.REG_UNKNOWN);
```

> **Note:** This procedure is overloaded. The `snapshot_id` and `flag` parameters are mutually exclusive.

Parameters

Table 8–301 REGISTER_SNAPSHOT Procedure Parameters

Parameter	Description
`sowner`	Owner of the snapshot.
`snapname`	Name of the snapshot.
`snapsite`	Name of the snapshot site for a snapshot registering at an Oracle8 or greater master. This should not contain any double quotes.
`snapshot_id`	The identification number of the snapshot. Specify an Oracle8 snapshot as a `BINARY_INTEGER`; specify an Oracle7 snapshot registering at an Oracle8 or greater master sites as a `DATE`.
`flag`	PL/SQL package variable indicating whether subsequent create or move commands are registered in the query text.
`query_txt`	The first 32,000 bytes of the query.
`rep_type`	Version of the snapshot. Valid constants that can be assigned include `reg_uknown` (the default), `reg_v7_group`, `reg_v8_group`, and `reg_repapi_group`.

Usage Notes

This procedure is executed at the master site, and can be done by a remote procedure call. If `REGISTER_SNAPSHOT` is called multiple times with the same

SNAPOWNER, SNAPNAME, and SNAPSITE, then the most recent values for SNAPSHOT_ID, FLAG, and QUERY_TXT are stored. If a query exceeds the maximum VARCHAR2 size, then QUERY_TXT contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the values of SNAPSHOT_ID and FLAG have to be looked up in the snapshot views by the person who calls the procedure.

If you do *not* want the snapshot query registered at the master site, then call the SET_REGISTER_QUERY_TEXT procedure with the option set to FALSE. To see the most recent setting of the option, call the GET_REG_QUERY_TEXT_FLAG function at the snapshot site before issuing the DDL.

UNREGISTER_SNAPSHOT procedure

This procedure enables the administration of individual snapshots. Invoked at a master site to unregister a snapshot.

Syntax

```
DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT (
   snapowner      IN   VARCHAR2,
   snapname       IN   VARCHAR2,
   snapsite       IN   VARCHAR2);
```

Parameters

Table 8–302 UNREGISTER_SNAPSHOT Procedure Parameters

Parameters	Description
snapowner	Owner of the snapshot.
snapname	Name of the snapshot.
snapsite	Name of the snapshot site.

9

Data Dictionary Views

This chapter describes data dictionary views that can be useful to users of the advanced replication facility. The views are alphabetized within the following categories:

- Replication Catalog Views.
- Deferred Transaction Views.
- Snapshots and Snapshot Refresh Group Views.

Replication Catalog Views

Whenever you install advanced replication capabilities at a site, Oracle installs the replication catalog, which consists of tables and views, at that site. As shown in Table 9–1, the views are used by master and snapshot sites to determine such information as what objects are being replicated, where they are being replicated, and if any errors have occurred during replication. *You should not modify the replication catalog tables directly; use the procedures provided in the DBMS_REPCAT package.*

Each view has three versions, which have different prefixes: USER_*, ALL_*, and SYS.DBA_* unless otherwise stated. This section ignores any differences among these views.

This section contains information about the following views:

- REPGROUP View
- REPCATLOG View
- REPCAT_REFRESH_TEMPLATES View
- REPCAT_TEMPLATE_OBJECTS View
- REPCAT_TEMPLATE_PARMS View
- REPCAT_TEMPLATE_SITES View
- REPCAT_USER_AUTHORIZATIONS View
- REPCAT_USER_PARM_VALUES View
- REPCOLUMN View
- REPCOLUMN_GROUP View
- REPCONFLICT View
- REPDDL View
- REPGROUP_PRIVILEGES View
- REPGROUPED_COLUMN View
- REPKEY_COLUMNS View
- REPOBJECT View
- REPPARAMETER_COLUMN View
- REPPRIORITY View

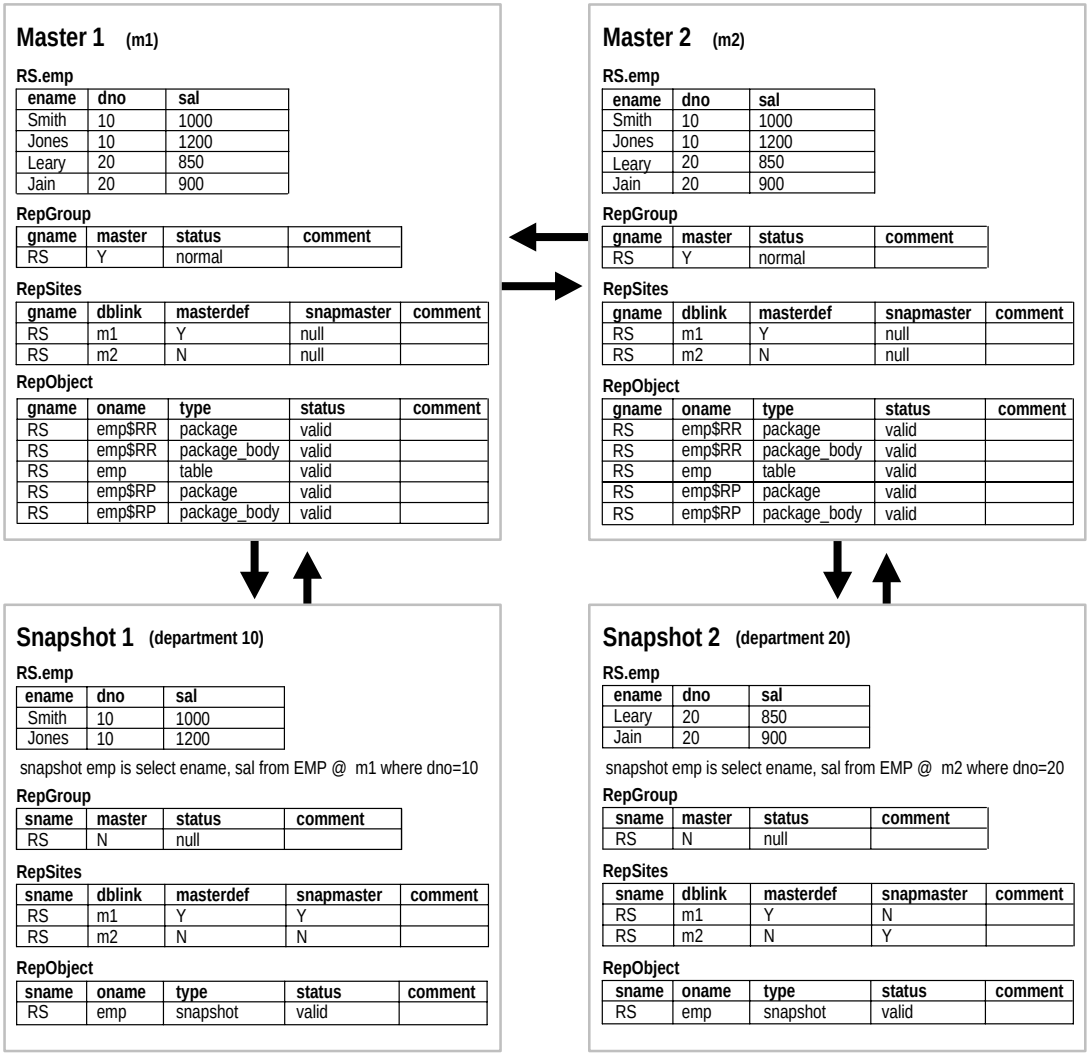
REPGROUP View

The REPGROUP view lists all of the object groups that are being replicated. The members of each object group are listed in a different view, REPOBJECT.

Table 9–1 REPGROUP View

Column	Description
sname	The name of the replicated schema. Obsolete with release 7.3 or later.
gname	The name of the replicated object group.
master	'Y' indicates that this is a master site. 'N' indicates the current site is a snapshot site.
status	Used at master sites only. Status can be: normal, quiescing, or quiesced.
schema_comment	Any user-supplied comments.
fname	Flavor name.
rpc_processing_disabled	'N' indicates that this site can receive and apply deferred RPCs. 'Y' indicates that this site can NOT receive and apply deferred RPCs.

Figure 9–1 Replication Catalog Views



REPCATLOG View

The REPCATLOG at each master site contains the interim status of any asynchronous administrative requests and any error messages generated. All messages encountered while executing a request are eventually transferred to the REPCATLOG at the master that originated the request. If an administrative request completes without error, ultimately all traces of this request are removed from the REPCATLOG view.

Table 9–2 REPCATLOG View

Column	Description
id	A sequence number. Together, the ID and SOURCE columns identify all log records at all master sites that pertain to a single administrative request.
source	Location that the request originated.
userid	Name of the user making the request.
timestamp	When the request was made.
role	Indicates if site is the 'masterdef' or a 'master' site.
master	If the role is 'masterdef' and the task is remote, indicates which master is performing the task.
sname	The name of the schema for the replicated object, if applicable.
request	The name of the DBMS_REPCAT administrative procedure that was run.
oname	The name of the replicated object, if applicable.
type	The type of replicated object.
status	The status of the administrative request: ready, do_ callback, await_callback, or error.
message	Any error message that has been returned.
errnum	The Oracle error number for the message.
gname	The name of the replicated object group.

REPCAT_REFRESH_TEMPLATES View

This view contains global information about the template, such as the template name, template owner, what refresh group the template objects will belong to, and the type of template (private vs. public).

When the DBA adds snapshot definitions to the template container, the DBA will reference the appropriate refresh_template_name. Any snapshots added to a specific template will be added to the refresh group specified in the REFRESH_GROUP_NAME parameter.

Furthermore, deployment templates created with the public parameter will be available to all users who can connect to the master site. Deployment templates created as private will be limited to those users listed in the DBA_REPCAT_USER_AUTHORIZATIONS table.

Table 9–3 REPCAT_REFRESH_TEMPLATES View

Column	Description
refresh_template_name	Name of the deployment template.
owner	Owner of the deployment template.
refresh_group_name	Name of the refresh group that template objects will be added to during the instantiation process.
template_comment	User supplied comment.
public_template	'Y' if the deployment template is public. 'N' if the deployment template is private.

REPCAT_TEMPLATE_OBJECTS View

The DBA_REPCAT_TEMPLATE_OBJECTS view contains the individual object definitions that are contained in a deployment template. Individual objects are added to a template by specifying the target template in the REFRESH_TEMPLATE_NAME parameter.

DDL_TEXT can contain variables to create parameterized templates. Variables are created by placing an ampersand (&) at the beginning of the variable name (i.e. &region). Parameterized templates allow for greater flexibility during the template instantiation process (i.e. in defining data sets specific for a snapshot site).

When the object is added to the template, the specified DDL is examined and if any parameters have been defined, Oracle will automatically add the parameter to the DBA_REPCAT_TEMPLATE_PARMS table.

Table 9–4 REPCAT_TEMPLATE_OBJECTS View

Column	Description
refresh_template_name	The name of the deployment template.
object_name	The name of the deployment template object.
object_type	The object type of the deployment template object.
ddl_text	The DDL that is executed to create the deployment template object. Object DDL is stored as a CLOB (see the notes below).
master_rollback_seg	The name of the rollback segment that is used during the instantiation of the deployment template object.
flavor_id	The flavor ID of the deployment template object.
derived_from_sname	If applicable, displays the schema that contains the object that the template object was created from.
derived_from_oname	If applicable, displays the name of the object that the template object was created from.

Note: Since the ddl_text parameter is defined as a CLOB, you will receive an error if you simply try to perform a SELECT on the DBA_REPCAT_TEMPLATE_OBJECTS view. If you don't need to see the object DDL, use the following select statement (be sure to exclude the `ddl_text` parameter):

```
SELECT refresh_template_name, object_name, object_type, master_rollback_seg,
flavor_id FROM dba_repcat_template_objects;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the DBA_REPCAT_TEMPLATE_OBJECTS view. For more information on using cursors see *Oracle8i Application Developer's Guide - Fundamentals*. For more information on using the DBMS_LOB package and LOBs in general, also see *Oracle8i Application Developer's Guide - Fundamentals*.

Use the following script to view the entire contents of DBA_REPCAT_TEMPLATE_OBJECTS view, including the `ddl_text` column:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, object_name, object_type, ddl_text,
          master_rollback_seg, flavor_id
          FROM dba_repcat_template_objects;
  tempstring VARCHAR2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.ddl_text);
        DBMS_LOB.READ(myrec.ddl_text, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.object_name||' '||myrec.object_type||' '||tempstring||' '||
          myrec.master_rollback_seg||' '||myrec.flavor_id);
  END LOOP;
END;
/
```

REPCAT_TEMPLATE_PARMS View

Parameters defined in the object DDL (see above) for a specific template are stored in the DBA_REPCAT_TEMPLATE_PARMS table. As previously stated, when an object is added to a template, the DDL is examined for variables; any found parameters are automatically added to this view.

You can also define default parameter values and a prompt string in this view (these make the templates more user friendly during the instantiation process).

Table 9–5 REPCAT_TEMPLATE_PARMS View

Column	Description
refresh_template_name	The name of the deployment template.
owner	The owner of the deployment template.
refresh_group_name	Name of the refresh group that template objects will be added to during the instantiation process.
template_comment	User specified comments.
public_template	'Y' if the deployment template is public. 'N' if the deployment template is private.

Table 9–5 REPCAT_TEMPLATE_PARMS View

Column	Description
parameter_name	The name of the parameter.
default_parm_value	The default parameter value. The `default_parm_value` is stored as a CLOB (see the notes below).
prompt_string	The prompt string for the parameter (under certain conditions, this prompt string will be displayed to the user when requesting a parameter value).
user_override	'Y' if the user can override the default parameter value. 'N' if the user can not override the default parameter value.

Note: Since the `default_parm_value` parameter is defined as a CLOB, you will receive an error if you simply try to perform a SELECT on the DBA_REPCAT_TEMPLATE_PARMS view. If you don't need to see the default parameter value, use the following select statement (be sure to exclude the `default_parm_value` parameter):

```
SELECT refresh_template_name, owner, refresh_group_name, template_comment,
public_template, parameter_name, prompt_string, user_override FROM dba_repcat_
template_parms;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the DBA_REPCAT_TEMPLATE_PARMS view. For more information on using cursors see *Oracle8i Application Developer's Guide - Fundamentals*. For more information on using the DBMS_LOB package and LOBs in general, also see *Oracle8i Application Developer's Guide - Fundamentals*.

Use the following script to view the entire contents of DBA_REPCAT_TEMPLATE_PARMS view, including the `default_parm_value` column:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, owner, refresh_group_name,
          template_comment, public_template, parameter_name, default_parm_value,
          prompt_string, user_override
          FROM dba_repcat_template_parms;
  tempstring VARCHAR2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.default_parm_value);
        DBMS_LOB.READ(myrec.default_parm_value, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.owner||' '||myrec.refresh_group_name||' '||
          myrec.template_comment||' '||myrec.public_template||' '||
          myrec.parameter_name||' '||tempstring||' '||myrec.prompt_string||' '||
          myrec.user_override);
  END LOOP;
END;
/
```

REPCAT_TEMPLATE_SITES View

The DBA_REPCAT_TEMPLATE_SITES view provides the DBA with information about the current status of template instantiation amongst the sites of a enterprise network. Specifically, the DBA will be able to monitor the installation and deletion of templates at specific sites.

Table 9–6 REPCAT_TEMPLATE_SITES View

Column	Description
refresh_template_name	The name of the deployment template.
refresh_group_name	Name of the refresh group that template objects will be added to during the instantiation process.
template_owner	Name of the user that is considered the owner of the deployment template.
user_name	The name of the user that instantiated the deployment template.
site_name	Target snapshot site of the deployment template.
repapi_site_id	The ID of the REPAPI site that has instantiated the displayed deployment template.
status	Displays the status of the deployment template at the target snapshot site: 0 = Not Installed 1 = Installed -1 = Installed with errors

REPCAT_USER_AUTHORIZATIONS View

If a template has been specified for private use, the authorized user list is displayed in the DBA_REPCAT_USER_AUTHORIZATIONS view. Users contained in this view will have the ability to instantiate the specified table, while users not on the list will not be able to instantiate the template.

Table 9–7 REPCAT_USER_AUTHORIZATIONS View

Column	Description
refresh_template_name	The name of the deployment template that a user has been authorized to instantiate.
owner	The name of the owner of the deployment template.
refresh_group_name	Name of the refresh group that template objects will be added to during the instantiation process.
template_comment	User specified comment.
public_template	'Y' if the deployment template is public. 'N' if the deployment template is private.
user_name	The name of the user that has been authorized to instantiate the specified deployment template.

REPCAT_USER_PARM_VALUES View

The DBA has the option of building a table of user parameters prior to distributing the template for instantiation. When a template is instantiated by a specified user, the values stored in the DBA_REPCAT_USER_PARM_VALUES table for the specified user will automatically be used.

Table 9–8 REPCAT_USER_PARM_VALUES View

Column	Description
refresh_template_name	The name of the deployment template that a user parameter value has been defined for.
owner	The name of the owner of the deployment template.
refresh_group_name	Name of the refresh group that the template objects will be added to during the instantiation process.
template_comment	User specified comment.
public_template	'Y' if the deployment template is public. 'N' if the deployment template is private.
parameter_name	The name of the parameter that a user parameter value has been defined for.
default_parm_value	The default value for the parameter. The `default_parm_value` is stored as a CLOB (see the notes below).
prompt_string	The prompt string for the parameter.
parm_value	The parameter value that has been defined for the specified user. The `parm_value` is stored as a CLOB (see the notes below).
user_name	The name of the user that the specified parameter value has been defined for.

Note: Since the `default_parm_value` and the `parm_value` parameters are defined as CLOBs, you will receive an error if you simply try to perform a SELECT on the DBA_REPCAT_USER_PARM_VALUES view. If you don't need to see the default or user parameter values, use the following select statement (be sure to exclude the `default_parm_value` and the `parm_value` parameters):

```
SELECT refresh_template_name, owner, refresh_group_name, template_comment,
public_template, parameter_name, prompt_string, user_name FROM dba_repcat_user_
parm_values;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the DBA_REPCAT_USER_PARM_VALUES view. For more information on using cursors see *Oracle8i Application Developer's Guide - Fundamentals*. For more information on using the DBMS_LOB package and LOBs in general, also see *Oracle8i Application Developer's Guide - Fundamentals*.

Use the following script to view the entire contents of DBA_REPCAT_TEMPLATE_PARMS view, including the `default_parm_value` column:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, owner, refresh_group_name,
          template_comment, public_template, parameter_name, default_parm_value,
          prompt_string, parm_value, user_name
          FROM dba_repcat_user_parm_values;
  tempstring VARCHAR2(1000);
  tempstring2 varchar2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.default_parm_value);
        DBMS_LOB.READ(myrec.default_parm_value, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.owner||' '||myrec.refresh_group_name||' '||
          myrec.template_comment||' '||myrec.public_template||' '||
          myrec.parameter_name||' '||tempstring||' '||myrec.prompt_string||' '||
          tempstring2||' '||myrec.user_name);
  END LOOP;
END;
/
```

REPCOLUMN View

The REPCOLUMN view lists the replicated columns for a table.

Table 9–9 REPCOLUMN View

Column	Description
sname	The name of the object owner.
oname	The name of the object.
type	The type of the object.
cname	The name of the replicated column.
id	The id of the replicated column.
pos	The ordering of the replicated column.
compare_old_on_delete	Indicates whether Oracle compares the old value of the column in replicated deletes.
compare_old_on_update	Indicates whether Oracle compares the old value of the column in replicated updates.
send_old_on_delete	Indicates whether Oracle sends the old value of the column in replicated deletes.
send_old_on_update	Indicates whether Oracle sends the old value of the column in replicated updates.
ctype	Displays the column type.
data_length	Displays the length of the column in bytes.
data_precision	Displays the column precision in terms of decimal digits for NUMBER columns or binary digits for FLOAT columns.
data_scale	Displays the digits to right of decimal point in a number.
nullable	Indicates if the column allow NULL values.
character_set_name	If applicable, displays the name of character set for the column.

REPCOLUMN_GROUP View

The REPCOLUMN_GROUP view lists all of the column groups that you have defined for each replicated table.

Table 9–10 REPCOLUMN_GROUP View

Column	Description
sname	The name of the schema containing the replicated table.
oname	The name of the replicated table.
group_name	The column group name.
group_Comment	Any user-supplied Comments.

Note: The sname column is not present in the USER_ view.

REPCONFLICT View

The REPCONFLICT view displays the name of the table for which you have defined a conflict resolution method and the type of conflict that the method is used to resolve.

Table 9–11 REPCONFLICT View

Column	Description
sname	The name of the schema containing the replicated table.
oname	The name of the table for which you have defined a conflict resolution method.
conflict_type	The type of conflict that the conflict resolution method is used to resolve: delete, uniqueness, or update.
reference_name	The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.

Note: The sname column is not present in the USER_ view.

REPDDL View

The REPDDL holds DDL for replication objects.

Table 9–12 REPDDL View

Column	Description
log_id	Identifying number of the REPCAT log record.
source	Name of the database at which the request originated.
role	'Y' if this database is the masterdef for the request; 'N' if this database is a master.
master	Name of the database that processes this request.
line	Ordering of records within a single request.
text	Portion of an argument or DDL text.

REPGROUP_PRIVILEGES View

The REPGROUP_PRIVILEGES view lists information about users who are registered for object group privileges.

Table 9–13 REPGENERATED View

Column	Description
username	Displays the name of the user.
gname	Displays the name of the replicated object group.
created	Displays the date that the object group was registered.
receiver	Indicates whether the user has receiver privileges.
proxy_snapadmin	Indicates whether the user has proxy_snapadmin privileges.

REPGROUPED_COLUMN View

The REPGROUPED_COLUMN view lists all of the columns that make up the column groups for each table.

Table 9–14 REPGROUPED_COLUMN View

Column	Description
sname	The name of the schema containing the replicated table.
oname	The name of the replicated table.
group_name	The name of the column group.
column_name	The name of the column in the column group.

Note: The sname column is not present in the USER_ version of the view.

REPKEY_COLUMNS View

The REPKEY_COLUMNS view lists information relating to the primary key column.

Table 9–15 REPKEY_COLUMNS View

Column	Description
sname	Owner of the replicated table.
oname	Name of the replicated table.
col	"Primary Key" column name in the table.

REPOBJECT View

The REPOBJECT view provides information about the objects in each replicated object group. An object can belong to only one object group. A replicated object group can span multiple schemas.

Table 9–16 REPOBJECT View

Column	Description
sname	The name of the schema containing the replicated object.
oname	The name of the replicated object.
type	The type of replicated object: table, view, package, package body, procedure, function, index, synonym, trigger, or snapshot.
status	CREATE indicates that Oracle is applying user supplied or Oracle-generated DDL to the local database in an attempt to create the object locally. When a local replica exists, Oracle COMPAREs the replica with the master definition to ensure that they are consistent. When creation or comparison complete successfully, Oracle updates the status to VALID; otherwise, it updates the status to ERROR. If you drop an object, Oracle updates its status to DROPPED before deleting the row from the REPOBJECT view.
id	The identifier of the local database object, if one exists.
object_Comment	Any user supplied Comments.
gname	The name of the replicated object group to which the object belongs.
generation_status	Status of whether the object needs to generate replication packages.
min_communication	N = send both OLD and NEW values for an updated view.
internal_package_exists	Y = internal package exists N = internal package does not exist
replication_trigger_exists	Y = internal replication trigger exists N = internal replication trigger does not exist

REPPARAMETER_COLUMN View

In addition to the information contained in the REPRESOLUTION view, the REPPARAMETER_COLUMN view also contains information about the columns that you indicated should be used to resolve the conflict. These are the column values that are passed as the LIST_OF_COLUMN_NAMES argument to the ADD_*_RESOLUTION procedures in the DBMS_REPCAT package.

Table 9–17 REPPARAMETER_COLUMN View

Column	Description
sname	The name of the schema containing the replicated table.
oname	The name of the replicated table.
conflict_type	The type of conflict that the routine is used to resolve: delete, uniqueness, or update.
reference_name	The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
sequence_no	The order that resolution methods are applied, with 1 being applied first.
method_name	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
function_name	For methods of type 'user function', the name of the user-supplied conflict resolution routine.
priority_group	For methods of type 'priority group', the name of the priority group.
parameter_table_name	Displays the name of the table that the parameter column belongs to.
parameter_column_name	The name of the column used as the IN parameter for the conflict resolution routine.
parameter_sequence_no	Ordering of column used as IN parameter.

Note: The SNAME column is not present in the USER_ view.

REPPRIORITY View

The REPPRIORITY view displays the value and priority level of each priority group member. Priority group names must be unique within a replicated object group. Priority levels and values must each be unique within a given priority group.

Table 9–18 REPPRIORITY View

Column	Description
sname	The name of the replicated schema. Obsolete with release 7.3 or later.
gname	The name of the replicated object group.
priority_group	The name of the priority group or site priority group.
priority	The priority level of the member. The highest number has the highest priority.
data_type	The datatype of the values in the priority group.
fixed_data_length	The maximum length of values of datatype CHAR.
char_value	The value of the priority group member, if data_type = char.
varchar2_value	The value of the priority group member, if data_type = varchar2.
number_value	The value of the priority group member, if data_type = number.
date_value	The value of the priority group member, if data_type = date.
raw_value	The value of the priority group member, if data_type = raw.
nchar_value	The value of the priority group member, if data_type = nchar.
nvarchar2_value	The value of the priority group member, if data_type = nvarchar2.
large_char_value	The value of the priority group member, for blank-padded character strings over 255 characters.

Note: The SNAME and GNAME columns are not present in the USER_ view.

REPPRIORITY_GROUP View

The REPPRIORITY_GROUP view lists the priority and site priority groups that you have defined for a replicated object group.

Table 9–19 REPPRIORITY_GROUP View

Column	Description
sname	The name of the replicated schema. Obsolete with release 7.3 or later. Not shown in USER views.
gname	The name of the replicated object group. Not shown in USER views.
priority_group	The name of the priority group or site priority group.
data_type	The datatype of the values in the priority group.
fixed_data_length	The maximum length for values of datatype CHAR.
priority_Comment	Any user-supplied Comments.

REPPROP View

The REPPROP view indicates the technique used to propagate operations on an object to the same object at another master site. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly.

Table 9–20 REPPROP View

Column	Description
sname	The name of the schema containing the replicated object.
oname	The name of the replicated object.
type	The type of object being replicated.
dblink	The fully qualified database name of the master site to which changes are being propagated.
how	How propagation is performed. Values recognized are 'none' for the local master site, and 'synchronous' or 'asynchronous' for all others.
propagate_ Comment	Any user-supplied Comments.

REPRESOLUTION View

The REPRESOLUTION view indicates the routines used to resolve update, unique or delete conflicts for each table replicated using row-level replication for a given schema.

Table 9–21 REPRESOLUTION View

Column	Description
sname	The name of the replicated schema.
oname	The name of the replicated table.
conflict_type	The type of conflict that the routine is used to resolve: delete, uniqueness, or update.
reference_name	The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
sequence_no	The order that resolution methods are applied, with 1 being applied first.
method_name	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
function_name	For methods of type 'user function', the name of the user-supplied conflict resolution routine.
priority_group	For methods of type 'priority group', the name of the priority group.
resolution_ Comment	Any user-supplied Comments.

Note: The SNAME column is not present in the USER_ view.

REPRESOL_STATS_CONTROL View

The REPRESOL_STATS_CONTROL view lists information about statistics collection for conflict resolutions for all replicated tables in the database.

Table 9–22 REPRESOL_STATS_CONTROL View

Column	Description
sname	Owner of the table.
oname	Table name.
created	Timestamp for which statistics collection was first started.
status	Status of statistics collection: ACTIVE, CANCELLED.
status_update_date	Timestamp for which the status was last updated.
purged_date	Timestamp for the last purge of statistics data.
last_purged_start_date	The last start date of the statistics purging date range.
last_purged_end_date	The last end date of the statistics purging date range.

Note: The SNAME column is not present in the USER_ view.

REPRESOLUTION_METHOD View

The REPRESOLUTION_METHOD view lists all of the conflict resolution routines available in your current database. Initially, this view lists the standard routines provided with the advanced replication facility. As you create new user functions and add them as conflict resolution methods for an object in the database, these functions are added to this view.

Table 9–23 REPRESOLUTION_METHOD View

Column	Description
conflict_type	The type of conflict that the resolution routine is designed to resolve: update, uniqueness, or delete.
method_name	The name of the Oracle-supplied method, or the name of the user-supplied routine.

REPRESOLUTION_STATISTICS View

The REPRESOLUTION_STATISTICS view lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables. These statistics are only gathered for a table if you have called DBMS_REPCAT.REGISTER_STATISTICS.

Table 9–24 REPRESOLUTION_STATISTICS View

Column	Description
sname	The name of the replicated schema.
oname	The name of the replicated table.
conflict_type	The type of conflict that was successfully resolved: delete, uniqueness, or update.
reference_name	The object to which the conflict resolution routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
method_name	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
function_name	For methods of type 'user function', the name of the user supplied conflict resolution routine.
priority_group	For methods of type 'priority group', the name of the priority group.
primary_key_value	A concatenated representation of the row's primary key.
resolved_date	Date on which the conflict for this row was resolved.

Note: The SNAME column is not present in the USER_ view.

REPSITES View

The REPSITES view lists the members of each replicated object group.

Table 9–25 Columns in USER, ALL, and DBA REPSITES View

Column	Description
gname	The name of the replicated object group.
dblink	The database link to a master site for this object group.
masterdef	Indicates which of the dblinks is the master definition site.
snapmaster	Used by snapshot sites to indicate which of the dblinks to use when refreshing.
master_Comment	User-supplied Comments.
master	Is the site a master site for the replicated group? Y or N.

The DBA_REPSITES view has the following additional columns:

prop_updates	Encoding of propagating technique for master.
my_dblink	Used to detect problem after import. If Y, the dblink is the global name.

REPGENOBJECTS View

The REPGENOBJECTS view describes objects generated to support replication.

Table 9–26 REPGENOBJECTS View

Column	Description
Base_oname	The object for which this object was generated.
Base_sname	The base object's owner.
Base type	The type of the base object.
Distributed	Obsolete.
Oname	The name of the generated object.
Package-prefix	The prefix for the package wrapper.
Procedure-prefix	The procedure prefix for the package wrapper.
Reason	The reason the object was generated.
Sname	The name of the replicated schema.
Type	The type of the generated object.

Deferred Transaction Views

Oracle provides several views for you to use in administering deferred transactions. These views provide information about each deferred transaction, such as the transaction destinations, the deferred calls that make up the transactions, and any errors encountered during attempted execution of the transaction. *You should not modify the tables directly; use the procedures provided in the DBMS_DEFER and DBMS_DEFER_SYS packages.*

DEFCALL View

The DEFCALL view records all deferred remote procedure calls.

Table 9–27 DEFCALL View

Column	Description
callno	Unique ID of call within a transaction.
deferred_tran_id	The unique ID of the associated transaction.
schemaname	The schema name of the deferred call.
packagename	The package name of the deferred call. For a replicated table, this may refer to the table name.
procname	The procedure name of the deferred call. For a replicated table, this may refer to an operation name.
argcount	The number of arguments to the deferred call.

DEFCALLDEST View

The DEFCALLDEST view lists the destinations for each deferred remote procedure call.

Table 9–28 DEFCALLDEST View

Column	Description
callno	Unique ID of call within a transaction.
deferred_tran_id	Corresponds to the deferred_tran_id in the DEFTRAN view. Each deferred transaction is made up of one or more deferred calls.
dblink	The fully qualified database name of the destination database.

DEFDEFAULTDEST View

If you are not using Oracle's replication facility and do not supply a destination for a deferred transaction or the calls within that transaction, Oracle uses the DEFDEFAULTDEST view to determine the destination databases to which you want to defer a remote procedure call.

Table 9–29 DEFDEFAULTDEST View

Column	Description
dblink	The fully qualified database name to which to replicate a transaction.

DEFERRCOUNT View

The DEFERRCOUNT view provides information about the error transactions for a given destination.

Table 9–30 DEFERRCOUNT View

Column	Description
errcount	Number of existing transactions that caused an error for the destination.
destination	Database link used to address destination.

DEFERROR View

The DEFERROR view provides the ID of each transaction that could not be applied. You can use this ID to locate the queued calls associated with this transaction. These calls are stored in the DEFCALL view. You can use the procedures in the DBMS_ DEFER_QUERY package to determine the arguments to the procedures listed in the DEFCALL view.

Table 9–31 DEFERROR View

Column	Description
deferred_tran_id	The ID of the transaction causing the error.
callno	Unique ID of call at deferred_tran_db.
destination	Database link used to address destination.
error_number	Oracle error number.
error_msg	Error message text.
receiver	Original receiver of the deferred transaction.
origin_tran_db	The database originating the deferred transaction.
origin_tran_id	The original ID of the transaction.
start_time	Time the original transaction was enqueued.

DEFLOB View

The DEFLOB view stores the LOB parameters to deferred RPCs.

Table 9–32 DEFLOB View

Column	Description
id	Identifier of the LOB parameter.
deferred_tran_id	Trnsaction ID for deferred RPC with this LOB parameter.
blob_col	The binary LOB parameter.
clob_col	The character LOB parameter.
nclob_col	The national character LOB parameter.

DEFPROPAGATOR View

The DEFPROPAGATOR view displays information about the local propagator.

Table 9–33 DEFPROPAGATOR View

Column	Description
username	Username of the propagator.
userid	User ID of the propagator.
status	Status of the propagator.
created	Time when the propagator was registered.

DEFSCHEDULE View

The DEFSCHEDULE view displays information about when a job is next scheduled to be executed.

Table 9–34 DEFSCHEDULE View

Column	Description
dblink	Fully qualified pathname to master database site for which you have scheduled periodic execution of deferred remote procedure calls.
job	Number assigned to job when you created it by calling DBMS_DEFER_SYS.SCHEDULE_PUSH. Query the WHAT column of USER_JOBS view to determine what is executed when the job is run.
interval	Function used to calculate the next time to push the deferred transaction queue to destination.
next_date	Next date that job is scheduled to be executed.
last_date	Last time the queue was pushed (or attempted to push) remote procedure calls to this destination.
disabled	Is propagation to destination disabled?
last_txn_count	Number of transactions pushed during last attempt.
last_error_number	Oracle error number from last push.
last_error_message	Error message from last push.

DEFTRAN View

The DEFTRAN view records all deferred transactions.

Table 9–35 DEFTRAN View

Column	Description
deferred_tran_id	The transaction ID that enqueued the calls.
delivery_order	An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating transaction.
destination_list	'R' or 'D'. 'R' indicates that the destinations are determined by the REPSITES view. 'D' indicates that the destinations were determined by the DEFDEFAULTDEST view or the NODE_LIST argument to the TRANSACTION or CALL procedures.
start_time	The time that the original transaction was enqueued.

DEFTRANDEST View

The DEFTRANDEST view lists the destinations for a deferred transaction.

Table 9–36 DEFTRANDEST View

Column	Description
deferred_tran_id	The transaction ID of the transaction to replicate to the given database link.
dblink	The fully qualified database name of the destination database.
delivery_order	An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating transaction.

Snapshots and Snapshot Refresh Group Views

The following views provide information about snapshots and snapshot refresh groups.

SNAPSHOTS View

The SNAPSHOTS catalog view lists information about all of the snapshots in a database.

Table 9–37 SNAPSHOTS View

Column	Description
OWNER	Owner of the snapshot.
NAME	Name of the view used by users and applications for querying and updating the snapshot.
TABLE_NAME	Table in which the snapshot is stored (it has an extra column for the master rowid).
MASTER_VIEW	View of the master table, owned by the snapshot owner, used for refreshes.
MASTER_OWNER	Owner of the master table.
MASTER	Master table that this snapshot copies.
MASTER_LINK	Database link name to the master site.
CAN_USE_LOG	YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log.
UPDATABLE	'YES' indicates snapshot is updatable; 'NO' indicates read-only.
LAST_REFRESH	Date and time at the master site of the last refresh.
ERROR	Number of failed attempts since last successful attempt.

Table 9–37 SNAPSHOTS View

Column	Description
TYPE	Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE.
NEXT	Date function used to compute next refresh dates.
START_WITH	Date function used to compute next refresh dates.
REFRESH_METHOD	Values used to drive a fast or complete refresh of the snapshot.
FR_OPERATIONS	If 'REGENERATE', then fast refresh operations have not been generated.
CR_OPERATIONS	If 'REGENERATE', then complete refresh operations have not been generated.
MASTER_ROLLBACK_SEG	Rollback segment to be used during refresh at the master.
REFRESH_GROUP	Group identifier for consistent refresh.
UPDATE_TRIG	Name of the trigger that fills in the UPDATE_LOG for an updatable snapshot.
UPDATE_LOG	Name of the table that logs changes to an updatable snapshot.
QUERY	Query used to create the snapshot.
STATUS	Displays the status of the snapshot. Valid values are: VALID: Snapshot is a read-consistent replica of the target master table from a specific point-in-time. INVALID: Snapshot is NOT a read-consistent replica of the target master table from a specific point-in-time. UNKNOWN: The read-consistent status of the snapshot is not known.
REFRESH_MODE	Displays how the snapshot will be refreshed. Valid values are PERIODIC and DEMAND.

Note: UPDATE_TRIG: NULL in Oracle8 or greater because of internalized triggers; MASTER_VIEW: NULL in Oracle8 or greater, now obsolete.

REGISTERED_SNAPSHOTS View

This view describes local or remote snapshots of local tables.

Table 9–38 REGISTERED_SNAPSHOTS View

Column	Description
OWNER	Owner of the snapshot.
NAME	The name of the snapshot.
SNAPSHOT_SITE	Global name of the snapshot site.
CAN_USE_LOG	If NO, the snapshot is complex and cannot fast refresh.
UPDATABLE	If NO, the snapshot is read only.
REFRESH_METHOD	Whether the snapshot uses ROWIDs or primary key for fast refresh.
SNAPSHOT_ID	Identifier for the snapshot used by the master for fast refresh.
QUERY_TXT	Query defining the snapshot, if registered.
VERSION	Version of the snapshot.

SNAPSHOT_LOGS View

The SNAPSHOT_LOGS view describes all the snapshot logs in the database.

Table 9–39 SNAPSHOT_LOGS View

Column	Description
LOG_OWNER	Owner of the snapshot log.
MASTER	Name of the master table for which changes are logged.
LOG_TABLE	Log table; with ROWIDs and timestamps of rows which changed in the master.
LOG_TRIGGER	An after-row trigger on the master which inserts rows into the log. NULL in Oracle 8 because of internalized triggers.
ROWIDS	If YES, the snapshot log records rowid information.
PRIMARY_KEY	If YES, the snapshot log records primary key information.

Table 9–39 SNAPSHOT_LOGS View

Column	Description
FILTER_COLUMNS	If YES, the snapshot log records filter column information.
CURRENT_SNAPSHOTS	One date per snapshot -- the date the snapshot of the master last refreshed.
SNAPSHOT_ID	Unique identifier of the snapshot.

Note: The view shows one row for each snapshot using the log. However, there is only one for all the snapshots located at the master site. To find out which logs are used, query USER_ SNAPSHOT_LOGS using unique and not selecting SNAPSHOT_ID and CURRENT_SNAPSHOTS.

SNAPSHOT_REFRESH_TIMES View

The REFRESH_TIMES view lists the date and time of the last refresh.

Table 9–40 SNAPSHOT_REFRESH _TIMES View

Column	Description
OWNER	Owner of the snapshot.
NAME	Name of the snapshot view.
MASTER_OWNER	Owner of the master table.
MASTER	Name of the master table.
LAST_REFRESH	The date and time of the last refresh.

REFRESH View

The REFRESH view lists each refresh group in the database, and describes refresh intervals for each group.

Table 9–41 REFRESH View

Column	Description
ROWNER	Owner of the refresh group.
RNAME	Name of the refresh group.
REFGROUP	ID number of the refresh group.
IMPLICIT_DESTROY	Implicit delete flag. If this value is Y, Oracle deletes the refresh group after you have subtracted the last member from the group.
JOB	ID number of the job used to execute the automatic refresh of the snapshot group. You can use this information to query the USER_JOBS view for more information about the job.
NEXT_DATE	Date when the members of the group will next be refreshed.
INTERVAL	The function used to calculate the interval between refreshes.
BROKEN	Flag used to indicate a problem with the refresh group. If the value of the broken flag is Y, Oracle will not refresh the group, even if it is scheduled to be refreshed.
PUSH_DEFERRED_RPC	If Y, push changes to master before refresh.
REFRESH_AFTER_ERRORS	If Y, proceed with refresh despite errors when pushing deferred RPCs.
ROLLBACK_SEG	Name of rollback segment used at the snapshot site during refresh.
PURGE_OPTION	The method for purging the transaction queue after each push.
PARALLELISM	The level of parallelism for transaction propagation.
HEAP_SIZE	The heap size used for transaction propagation.

REFRESH_CHILDREN View

The REFRESH_CHILDREN view lists the members of each refresh group owned by the user, and includes information about the refresh interval for each member.

Table 9–42 REFRESH_CHILDREN View

Column	Description
OWNER	Owner of the refresh group member.
NAME	Name of the refresh group member.
TYPE	Type of the refresh group member; for example, SNAPSHOT.
ROWNER	Owner of the refresh group.
RNAME	Name of the refresh group.
REFGROUP	ID number of the refresh group.
IMPLICIT_DESTROY	Implicit delete flag. If this value is Y, Oracle deletes the refresh group after you have subtracted the last member from the group.
JOB	ID number of the job used to execute the automatic refresh of the snapshot refresh group. You can use this information to query the USER_JOBS view for more information about the job.
NEXT_DATE	Date when the members of the group will next be refreshed.
INTERVAL	The function used to calculate the interval between refreshes.
BROKEN	Flag used to indicate a problem with the refresh group. If the value of the broken flag is Y, Oracle will not refresh the group, even if it is scheduled to be refreshed.
PUSH_DEFERRED_RPC	If Y, push changes to master before refresh.
REFRESH_AFTER_ERRORS	If Y, proceed with refresh despite errors when pushing deferred RPCs.
ROLLBACK_SEG	Name of rollback segment used at the snapshot site during refresh.
PURGE_OPTION	The method for purging the transaction queue after each push.
PARALLELISM	The level of parallelism for transaction propagation.
HEAP_SIZE	The heap size used for transaction propagation.

Index

A

C

D

E

F

G

I

J

L

M

O

P

Q

R

S

T

V