Oracle® Database

Backup and Recovery Advanced User's Guide

10*g* Release 1 (10.1)

Part No. B10734-01

December 2003

ORACLE

Oracle Database Backup and Recovery Advanced User's Guide, 10*g* Release 1 (10.1)

Part No. B10734-01

Primary Author: Antonio Romero

Contributing Author: Lance Ashdown

Contributors: Beldalker Anand, Tammy Bednar, Senad Dizdar, Muthu Olagappan, Francisco Sanchez, Steve Wertheimer

Graphic Artist: Valarie Moore

Contents

2 RMAN Backups Concepts

3 RMAN Recovery Concepts

6 Configuring the RMAN Environment: Advanced Topics

7 Making Backups with RMAN: Advanced Topics

8 Advanced RMAN Recovery Techniques

9 Flashback Technology: Recovering from Logical Corruptions

10 RMAN Tablespace Point-in-Time Recovery (TSPITR)

11 Duplicating a Database with Recovery Manager

12 Migrating Databases To and From ASM with Recovery Manager

13 Managing the Recovery Catalog

14 Tuning Backup and Recovery

15 Recovery Manager Troubleshooting

Part III Performing User-Managed Backup and Recovery

16 Making User-Managed Backups

17 Performing User-Managed Database Flashback and Recovery

18 Advanced User-Managed Recovery Scenarios

19 Performing User-Managed TSPITR

Index

Send Us Your Comments

Oracle Database Backup and Recovery Advanced User's Guide, 10*g* Release 1 (10.1)

Part No. B10734-01

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

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

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

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

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

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

Preface

This preface contains these topics:

Audience

Backup and Recovery Advanced User's Guide is intended for database administrators who perform the following tasks:

- Back up, restore, and recover Oracle databases
- Perform maintenance on backups of database files

To use this document, you need to know the following:

- Relational database concepts and basic database administration as described in *Oracle Database Concepts* and the *Oracle Database Administrator's Guide*
- Basic backup and recovery concepts and strategies as described in *Oracle Database Backup and Recovery Basics*
- The operating system environment under which you are running the database

Organization

This document contains:

Part I, "Recovery Manager Advanced Architecture and Concepts"

This section offers detailed conceptual information for Recovery Manager (RMAN).

Chapter 1, "Recovery Manager Architecture"

This chapter describes the application architecture of the RMAN environment.

Chapter 2, "RMAN Backups Concepts"

This chapter describes how to start RMAN and connect to target, catalog, and auxiliary databases.

Chapter 3, "RMAN Recovery Concepts"

This chapter describes basic concepts involved in RMAN restore, recovery, and database duplication.

Chapter 4, "RMAN Maintenance Concepts"

This chapter describes basic concepts involved in maintaining the RMAN repository.

Part II, "Performing Advanced RMAN Backup and Recovery"

This section describes advanced procedures for using RMAN.

Chapter 5, "Connecting to Databases with RMAN"

This chapter gives detailed information for how to connect to databases with RMAN.

Chapter 6, "Configuring the RMAN Environment: Advanced Topics"

This chapter describes advanced configurations in the RMAN environment.

Chapter 7, "Making Backups with RMAN: Advanced Topics"

This chapter describes detailed procedure for using the `BACKUP` command.

Chapter 8, "Advanced RMAN Recovery Techniques"

This chapter includes advanced scenarios and techniques using the `RESTORE` and `RECOVER` commands.

Chapter 9, "Flashback Technology: Recovering from Logical Corruptions"

This chapter describes the Flashback features of the Oracle database, and their use in a data recovery context.

Chapter 10, "RMAN Tablespace Point-in-Time Recovery (TSPITR)"

This chapter describes how to recover one or more tablespaces to a past point in time without affecting the rest of the database.

Chapter 11, "Duplicating a Database with Recovery Manager"

This chapter describes how to use `DUPLICATE` to create a copy of the target database.

Chapter 12, "Migrating Databases To and From ASM with Recovery Manager"

This chapter describes how to use RMAN to move databases into and out of Automatic Storage Management disk groups.

Chapter 13, "Managing the Recovery Catalog"

This chapter describes how to create and manage a recovery catalog.

Chapter 14, "Tuning Backup and Recovery"

This chapter gives tips for improving RMAN backup and restore performance.

Chapter 15, "Recovery Manager Troubleshooting"

This chapter gives tips for diagnosing and responding to RMAN problems.

Part III, "Performing User-Managed Backup and Recovery"

This section describes how to use operating system utilities to back up and restore a database and how to use the SQL*Plus RECOVER command.

Chapter 16, "Making User-Managed Backups"

This chapter describes how to use operating system command to back up database files and archived redo logs.

Chapter 17, "Performing User-Managed Database Flashback and Recovery"

This chapter describes how to use the SQL*Plus FLASHBACK DATABASE and RECOVER commands.

Chapter 18, "Advanced User-Managed Recovery Scenarios"

This chapter describes advanced scenarios involving user-managed restore and recovery.

Chapter 19, "Performing User-Managed TSPITR"

This chapter describes how to perform user-managed TSPITR.

Chapter 20, "Troubleshooting User-Managed Media Recovery"

This chapter describes how to diagnose and solve problems in user-managed media recovery.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Backup and Recovery Basics*
- *Oracle Database Recovery Manager Reference*
- *Oracle Database Utilities*
- `http://www.oracle.com/database/recovery`

You can access information about the Backup Solutions Program at

`http://otn.oracle.com/deploy/availability`

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

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Printed documentation is available for sale in the Oracle Store at

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

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

`http://otn.oracle.com/membership/`

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

`http://otn.oracle.com/documentation/`

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`* `[ , `*`precision`* `])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

...	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS `*`subquery`*`;`
		
		`SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`

.
.
.
	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;`
		`NAME`
		`----------`
		`/fsl/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fsl/dbs/tbs_09.dbf`
		`9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

Convention	Meaning	Example

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither

evaluates nor makes any representations regarding the accessibility of these Web sites.

What's New in Backup and Recovery?

This section describes new features of Recovery Manager in Oracle Database Release 10*g* and provides pointers to additional information. For information on new features in Oracle9*i* and previous releases, refer to the documentation for those releases.

Oracle Database Release 10*g* New Features in Backup and Recovery

The new features for this release greatly increase the manageability of RMAN, making backup and recovery simpler and more performant.

- Flash Recovery Area

  A flash recovery area is a directory, file system, or Automatic Storage Management disk group that serves as the default storage area for files related to recovery. Such files include

  – Multiplexed copies of the control file and online redo logs

  – Archived redo logs and flashback logs

  – RMAN backups

  – Files created by RESTORE and RECOVER commands

  Recovery components of the database interact with the flash recovery area to ensure that the database is completely recoverable using files in the flash recovery area. The database manages the disk space in the flash recovery area, and when there is not sufficient disk space to create new files, the database creates more room automatically by deleting the minimum set of files from flash recovery area that are obsolete, backed up to tertiary storage, or redundant.

  See Also:

  - *Oracle Database Backup and Recovery Basics* to learn how to set up the flash recovery area
  - "Configuring the Flash Recovery Area: Advanced Topics" on page 6-2

- Oracle Flashback Database

  With the FLASHBACK DATABASE command, you can quickly revert a database to a previous time--without restoring datafiles and performing media recovery. When you enable the Flashback Database feature, the database automatically creates, deletes, and manages flashback logs inside the flash recovery area. When you run the FLASHBACK DATABASE command, the database uses the flashback logs as well as the archived redo logs to reconstruct its contents at the specified time.

See Also: "Oracle Flashback Database: Alternative to Point-In-Time Recovery" on page 9-15

- Incrementally Updated Backups: Rolling Forward Image Copies

  You can apply backup incrementals to datafile image copies—not current datafiles—to roll them forward to a specified point in time. In this way, you can potentially reduce recovery time by applying incrementals to copies and avoid taking a full image copy again after incremental backups.

  See Also: *Oracle Database Backup and Recovery Basics* to learn how to roll forward image copies

- Disk Topology and Automatic Performance Tuning

  The new disk topology API extends RMAN's capability to more platforms and file types. RMAN is also able to tune its parameters automatically according to disk topology information, which decreases the degree of user intervention required for performance tuning. RMAN can automatically tune the following:

  – The contents of the backup sets that define which files are multiplexed
  – The level of multiplexing
  – The number and size of the input and output disk buffers

  See Also: "Tuning Recovery Manager: Overview" on page 14-2 to learn more about RMAN performance tuning

- Automatic Datafile Creation

  RMAN can create datafiles automatically when the user executes `RESTORE` or `RECOVER` commands in the following situations:

  – The control file has metadata for the datafile, that is, the control file was backed up after datafile creation, but the datafile itself is not backed up.
  – The control file does not have metadata for the datafile, that is, the user did not back up the control file after datafile creation.

  See Also: *Oracle Database Recovery Manager Reference* to learn about the behavior of the `RECOVER` command

- Recovery Through Resetlogs

  RMAN simplifies recovery with backups taken from an earlier incarnation so that it is as easy as recovering a backup from the same incarnation. Hence, you no longer need to make new backups of a database after a RESETLOGS. The procedure is as easy and transparent as recovering a backup from the same incarnation. Also, the ALTER DATABASE OPEN RESETLOGS statement is now modified so that the database now archives the current online redo logs (if possible) before clearing the logs.

  See Also: "Point-in-Time Recovery to a Previous Incarnation" on page 8-4

- Restore Failover

  If a backup piece is inaccessible or corrupted, then RMAN can automatically fail over to another copy of this backup piece during RESTORE. If all copies of this backup set are unusable, then RMAN can fail over to previous redundant backup sets. RMAN continously fails over to previous backups until it exhausts all possibilities. This feature is similar to archived log failover.

  See Also:

  - "Restore Failover" on page 3-4 for details of restore failover behavior.
  - *Oracle Database Recovery Manager Reference* to learn about the behavior of the RESTORE command

- BACKUP Command Creates Backup Sets or Image Copies

  In previous releases, RMAN had two separate commands to back up datafiles: BACKUP and COPY. The BACKUP command backed up the datafile into a backup set, which is a proprietary format that allows multiple datafiles to be multiplexed together. The COPY command generated image copies, that is, bit-by-bit copies of datafiles.

  Starting in Release 10*g*, the COPY command is deprecated in favor of an enhanced BACKUP command that enables you to specify whether RMAN should create copies or backup sets. As a result, BACKUP AS COPY can copy a database or multiple tablespaces, datafiles, archived logs and datafile copies.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to copy files
- *Oracle Database Recovery Manager Reference* to learn about the output of the BACKUP command

- Proxy Archived Log Backups

  RMAN can create and restore proxy backups of archived redo logs.

- Cataloging Backup Pieces

  Users can now catalog user-specified backup pieces with the CATALOG command. Cataloging a backup piece adds it to the RMAN repository so that it is available for use in recovery operations. This enhanced functionality is useful when you make a copy of a backup piece with an operating system utility, or when you move a backup piece from one disk to another so that it has a different absolute path name.

  See Also: *Oracle Database Backup and Recovery Basics* to learn how to catalog backup pieces, and *Oracle Database Recovery Manager Reference* for CATALOG syntax

- Fast Incremental Backups

  If you enable block change tracking, then the database automatically tracks which datafile blocks have changed in change tracking files. When you execute BACKUP INCREMENTAL, RMAN uses the change tracking file to more quickly identify the blocks changed since the previous incremental backup. As a result, RMAN creates incremental backups much faster than in prior releases.

  See Also:

  - *Oracle Database Backup and Recovery Basics* to learn how to enable block change tracking and manage the change tracking file

- Channel Failover

  If multiple channels are allocated for a BACKUP command, and if RMAN encounters a retriable error (for example, an unplanned instance shutdown in RAC, or a media management error), then RMAN attempts to move the backup to a different channel and complete the work.

See Also: "Channel Failover" on page 2-11

- Deferred Error Reporting

  For most RMAN commands, RMAN will report errors in the output when they occur and then continue to execute the command if possible. If RMAN can retry a job step on another channel, then it will report a message to this effect. If some job steps could not be completed, then the error stack at the end of command execution will display errors for failed steps.

- Improved RMAN Reporting Through V$ Views

  The V$RMAN_OUTPUT memory-only view shows the output of a currently executing RMAN job, whereas the V$RMAN_STATUS control file view indicates the status of both executing and completed RMAN jobs. The V$BACKUP_FILES provides access to the information used as the basis of the LIST BACKUP and REPORT OBSOLETE commands.

- Automatic Instance Creation for RMAN TSPITR

  RMAN can create an auxiliary instance automatically when you perform RMAN TSPITR. RMAN creates the auxiliary instance in the same machine as the target database. RMAN provides intelligent defaults for the instance, but you can provide nondefault initialization parameters if desired. RMAN automatically dismantles the auxiliary database and instance after a successful TSPITR.

  See Also: "RMAN Tablespace Point-in-Time Recovery (TSPITR)" on page 10-1

- Cross-Platform Tablespace Conversion

  The RMAN command CONVERT TABLESPACE enables you to transport a tablespace from a database running on one platform (for example, Solaris) to a database running on a different platform (for example, Windows).

  See Also: *Oracle Database Recovery Manager Reference* to learn about the CONVERT command

- Enhanced Stored Scripts Commands

  The recovery catalog now supports global stored scripts, which can be applied to any database in the recovery catalog. A number of new commands have been

added to allow for easier manipulation and displaying of stored scripts from the recovery catalog.

See Also: "Working with RMAN Stored Scripts in the Recovery Catalog" on page 13-15 for the new commands and options

- Binary Compression of Backup Sets

  RMAN can now write backup sets in a format that uses binary compression to reduce backup set size. Using compressed backup sets can save storage space, as well as network bandwidth when backing up across a network.

  See Also: *Oracle Database Backup and Recovery Basics* for more details on using compressed backupsets.

- Enhanced Reporting: RESTORE PREVIEW

  The PREVIEW option to the RESTORE command can now tell you which backups will be accessed during a RESTORE operation.

  See Also: *Oracle Database Backup and Recovery Basics* for more details on RESTORE PREVIEW

- Managing Backup Duration and Throttling

  The BACKUP command now accepts a DURATION clause, which lets you specify limited time windows for backup activities and minimize load imposed by backup activities during those backup windows. .

  See Also: "Managing Backup Windows and Performance: BACKUP... DURATION" on page 2-55 for more details on managing backup duration and throttling

Part I

Recovery Manager Advanced Architecture and Concepts

Part I describes the architecture of the RMAN environment and introduces basic concepts. This part contains these chapters:

- Chapter 1, "Recovery Manager Architecture"
- Chapter 2, "RMAN Backups Concepts"
- Chapter 3, "RMAN Recovery Concepts"
- Chapter 4, "RMAN Maintenance Concepts"

1

Recovery Manager Architecture

This chapter describes the Recovery Manager (RMAN) interface and the basic components of the RMAN environment.

This chapter contains these topics:

About the RMAN Environment

Recovery Manager (RMAN) is a client application that performs backup and recovery operations. The **Recovery Manager environment** consists of the various applications and databases that play a role in a backup and recovery strategy.

Table 1–1 lists possible components of the RMAN environment.

Table 1–1 Components of the RMAN Environment

Component	Description	Required?
Target database	The control files, datafiles, and optional archived redo logs that RMAN is in charge of backing up or restoring. RMAN uses the target database control file to gather metadata about the target database and to store information about its own operations. The work of backup and recovery is performed by server sessions running on the target database.	Yes
RMAN client	The client application that manages backup and recovery operations for a target database. The RMAN client can use Oracle Net to connect to a target database, so it can be located on any host that is connected to the target host through Oracle Net.	Yes
Recovery catalog database	A database containing the recovery catalog schema, which contains the metadata that RMAN uses to perform its backup and recovery operations.	No
Recovery catalog schema	The user within the recovery catalog database that owns the metadata tables maintained by RMAN. RMAN periodically propagates metadata from the target database control file into the recovery catalog.	No
Standby database	A copy of the primary database that is updated using archived logs created by the primary database. RMAN can create or back up a standby database. You can fail over to the standby database if the primary database goes down.	No
Duplicate database	A copy of the primary database that you can use for testing purposes.	No
Media management application	A vendor-specific application that allows RMAN to back up to a storage system such as tape.	No
Media management catalog	A vendor-specific repository of information about a media management application.	No

Table 1–1 Components of the RMAN Environment

Component	Description	Required?
Enterprise Manager	A browser-based interface to the Oracle database, including backup and recovery through RMAN.	No

The only required components in an RMAN environment are the target database and the RMAN client, but most real-world configurations are more complicated. One might use an RMAN client connecting to multiple media managers and multiple target, recovery catalog, and auxiliary databases, all accessed through Enterprise Manager.

RMAN Session Architecture

The RMAN client application directs database server sessions to perform all backup and recovery tasks. The meaning of "session" in this sense depends on the operating system. For example, on UNIX, a server session corresponds to a server process, while on Windows it corresponds to a thread within the database service.

The RMAN client itself does not perform backup, restore, or recovery operations. When you connect the RMAN client to a target database, RMAN allocates server sessions on the target instance and directs them to perform the operations. The RMAN client uses internal, undocumented PL/SQL packages to communicate with the target database and recovery catalog.

RMAN Command Line Client

Use the RMAN command line client to enter commands that you can use to manage all aspects of backup and recovery operations.

Even when you use the backup and recovery features in Enterprise Manager that are built on top of RMAN, an RMAN client executes behind the scenes.

Note: All RMAN commands for Oracle release 8.1 and higher also work in Oracle Databse Release 10*g*.

How RMAN Compiles and Executes Commands

RMAN processes most commands in the two phases discussed in this section:

Compilation Phase

During the compilation phase, RMAN determines which objects the command will access (for example, resolving a tablespace name into its component datafiles). Then, RMAN constructs a sequence of remote procedure calls (RPCs) that instruct the server sessions on the target database to perform the desired operation.

Execution Phase

During the execution phase, RMAN sends the RPC calls to the target database, monitors their progress, and collects the results. If more than one channel is allocated, then RMAN can execute certain commands in parallel so that all of the channels' target database sessions are concurrently executing an RPC call.

Issuing RMAN Commands

RMAN uses a command language interpreter (CLI) that can execute commands in interactive or batch mode.

Entering Commands at the RMAN Prompt

To run RMAN commands interactively, start RMAN and then type commands into the command-line interface. For example, you can start RMAN from the UNIX command shell and then execute interactive commands as follows:

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb
```

After the RMAN prompt is displayed, you can enter commands such as the following:

```
RMAN> BACKUP DATABASE;
```

Using RMAN with Command Files

A **command file** is a text file which contains RMAN commands as you would enter them at the command line. You can run the a command file by specifying its name on the command line. The contents of the command file will be interpreted as if entered at the command line. If the LOG command line argument is specified, RMAN directs output to the named log file. Command files are one way to automate scheduled backups through an operating system job control facility.

In this example, a sample RMAN script is placed into a command file called `commandfile.rcv`. You can run this file from the operating system command line and write the output into the log file `outfile.txt` as follows:

```
% rman TARGET / CATALOG rman/cat@catdb CMDFILE commandfile.rcv LOG outfile.txt
```

> **See Also:** *Oracle Database Recovery Manager Reference* for more information about RMAN command line options

Stored Scripts

A **stored script** is a block of RMAN job commands that is stored in the recovery catalog. By storing scripts in the recovery catalog, the script is available to any RMAN client that connects to the catalog and the target database. Stored scripts can be associated with a single database in the catalog, or they can be global stored scripts, which can be executed against any target database in the catalog.

> **See Also:** "Working with RMAN Stored Scripts in the Recovery Catalog" on page 13-15 for more on stored scripts. Also refer to the sample scripts in the `?/rdbms/demo` directory.

Commands Valid Only in RUN Blocks

There are RMAN commands which are only valid in RUN blocks. These typically involve setting up the environment within which the statements in the RUN block will execute. Typical of this group are `ALLOCATE CHANNEL` and `SET NEWNAME FOR DATAFILE`. Using these commands outside of a RUN block will generate an error.

> **See Also:** The syntax diagrams for the RUN command in *Oracle Database Recovery Manager Reference* regarding which commands are valid in RUN blocks

Commands Not Valid in RUN Blocks

There are a number of RMAN commands which cannot be used in RUN blocks. Typically these are used to control the RMAN environment (connecting to different databases, or configuring RMAN defaults), or to manage or query the recovery catalog (including creating and using stored scripts). Here are some examples:

- `CONNECT`
- `CONFIGURE`
- `CREATE CATALOG`, `DROP CATALOG`, `UPGRADE CATALOG`
- `CREATE SCRIPT`, `DELETE SCRIPT`, `REPLACE SCRIPT`
- `LIST`
- `REPORT`

You can include these commands inside command files, as long as they are not wrapped inside a RUN block. You cannot use them inside a stored script from the recovery catalog, because the contents of a stored script are executed within a RUN block.

> **See Also:** *T*he syntax diagrams for the RUN command in *Oracle Database Recovery Manager Reference* regarding which commands are valid in RUN blocks

Controlling RMAN Output

When you run RMAN in command line mode, it sends the output to the terminal. If you specify the LOG option, then RMAN writes the output to a specified log file instead.

Output for currently executing RMAN jobs is also stored in the V$RMAN_OUTPUT view, which reads only from memory (that is, the information is not stored in the control file). The V$RMAN_STATUS view stores metadata about jobs in progress as well as completed jobs. The metadata for completed jobs is stored in the control file.

RMAN Pipe Interface

The RMAN pipe interface is an alternative method for issuing commands to RMAN and receiving the output from those commands. With this interface, RMAN obtains commands and sends output by using the DBMS_PIPE PL/SQL package. RMAN does not read or write any data using the operating system shell. By using this interface, it is possible to write a portable programmatic interface to RMAN.

The pipe interface is invoked by using the PIPE command-line parameter. RMAN uses two private pipes: one for receiving commands and the other for sending output. The names of the pipes are derived from the value of the PIPE parameter. For example, you can invoke RMAN with the following command:

```
% rman PIPE abc TARGET SYS/oracle@trgt
```

RMAN opens the two pipes in the target database: ORA$RMAN_ABC_IN, which RMAN uses to receive user commands, and ORA$RMAN_ABC_OUT, which RMAN uses to send all output back to RMAN.

All messages on both the input and output pipes are of type VARCHAR2.

Note that RMAN does not permit the pipe interface to be used with public pipes, because they are a potential security problem. With a public pipe, any user who knows the name of the pipe can send commands to RMAN and intercept its output.

See Also: "Executing RMAN Commands Through a Pipe" on page 5-7 to learn how to execute RMAN commands through a pipe

RMAN Repository

The RMAN repository is the collection of metadata about the target databases that RMAN uses for backup, recovery, and maintenance. RMAN always stores this information in records in the control file. The version of this information in the control file is the authoritative record of RMAN's backups of your database. This is one reason why protecting your control file is a important part of your backup strategy. RMAN can conduct all necessary backup and recovery operations using just the control file to store the RMAN repository information, and maintains all records necessary to meet your configured retention policy.

You can also create a **recovery catalog**, an external Oracle database in which to store this information. The control file has finite space for records of backup activities, while a recovery catalog can store a much longer history. The added complexity of operating a recovery catalog database can be offset by the convenience of having the extended backup history available if you have to do a recovery that goes further back in time than the history in the control file.

There are also a few features of RMAN that only function when you use a recovery catalog. For example, RMAN stored scripts are stored in the recovery catalog, so commands related to them require the use of a recovery catalog. Other RMAN commands are specifically related to managing the recovery catalog and so are not available (and not needed) if RMAN is not connected to a recovery catalog.

The recovery catalog's version of the RMAN repository is maintained solely by RMAN. The target instance never accesses it directly. RMAN propagates information about the database structure, archived redo logs, backup sets, and datafile copies into the recovery catalog from the target database's control file after any operation that updates the repository, and also before certain operations.

See Also: *Oracle Database Backup and Recovery Basics* for details on how to manage the RMAN repository, and Chapter 13, "Managing the Recovery Catalog" to learn more about features specific to the recovery catalog

Storage of the RMAN Repository in the Recovery Catalog

It is recommended that you store the recovery catalog in a dedicated database. If you store the recovery catalog alongside other data in some other database, then if

you lose that other database you will lose your recovery catalog as well. This will make your recovery more difficult.

Registration of Databases in the Recovery Catalog

The enrolling of a database in a recovery catalog is called **registration**. You can register more than one target database in the same recovery catalog. For example, you can register databases `prod1`, `prod2`, and `prod3` in a single catalog owned by `catowner` in the database `catdb`. Because RMAN distinguishes databases by unique database identifier (DBID), each database registered in a given catalog must have a unique DBID.

> **See Also:** "Registering a Database in the Recovery Catalog" on page 13-5, and *Oracle Database Utilities* to learn how to use the DBNEWID utility to change the DBID of a database

Contents of the Recovery Catalog

The recovery catalog contains information about RMAN operations, including:

- Datafile and archived redo log backup sets and backup pieces
- Datafile copies
- Archived redo logs and their copies
- Tablespaces and datafiles on the target database
- Stored scripts, which are named user-created sequences of RMAN commands
- Persistent RMAN configuration settings

Resynchronization of the Recovery Catalog

The recovery catalog obtains crucial RMAN metadata from the target database control file. Resynchronization of the recovery catalog ensures that the metadata that RMAN obtains from the control file stays current. Resynchronizations can be full or partial.

> **See Also:** "Types of Records in the Control File" on page 1-10 for more information about control file records, and "When Should You Resynchronize?" on page 13-13

Snapshot Control File RMAN creates a **snapshot control file**, which is a temporary backup control file, in an operating system specific location each time it performs a full resynchronization. This snapshot control file ensures that RMAN has a

consistent view of the control file. Because the snapshot control file is intended for RMAN's short-term use, it is not registered in the recovery catalog. RMAN records the snapshot control file checkpoint in the recovery catalog to indicate the currency of the recovery catalog.

The database server ensures that only one RMAN session accesses a snapshot control file at any point in time. This safeguard is necessary to ensure that two RMAN sessions do not interfere with each other's use of the snapshot control file.

Note: You can specify the name and location of the snapshot control file. For instructions, refer to "Setting the Snapshot Control File Location" on page 6-26.

See Also: "Managing the Control File When You Use a Recovery Catalog" on page 13-21 to learn how to resynchronize the recovery catalog, and *Oracle Database Recovery Manager Reference* for syntax

Backups of the Recovery Catalog

A single recovery catalog is able to store information for multiple target databases. Consequently, loss of the recovery catalog can be disastrous. You should back up the recovery catalog frequently.

If the recovery catalog is destroyed and no backups of it are available, then you can partially reconstruct the catalog from the current control file or control file backups. Nevertheless, you should always aim to have a valid, recent backup of the catalog.

See Also: "Backing Up the Recovery Catalog" on page 13-22 to learn how to back up the recovery catalog

Compatibility of the Recovery Catalog

When you use RMAN with a recovery catalog in an environment where you have run past versions of the Oracle database, you can wind up with versions of the RMAN client, recovery catalog database, recovery catalog schema, and target database that all originated in different releases of the database. You will find a

compatibility matrix in *Oracle Database Recovery Manager Reference* that describes supported interoperability scenarios.

Storage of the RMAN Repository in the Control File

Because most information in the recovery catalog is also available in the target database's control file, RMAN supports an operational mode in which it uses the target database control file instead of a recovery catalog. This mode is especially appropriate for small databases where installation and administration of a separate recovery catalog database is burdensome. The only RMAN feature that is not supported in `NOCATALOG` mode is stored scripts.

Types of Records in the Control File

When you do not use a recovery catalog, the control file is the exclusive source of information about backups and copies as well as other relevant information. The control file contains two types of records: **circular reuse records** and **noncircular reuse records**.

Circular Reuse Records Circular reuse records contain noncritical information that is eligible to be overwritten if the need arises. These records contain information that is continually generated by the database. Circular reuse records are arranged in a logical ring. When all available record slots are full, the database either expands the control file to make room for a new record or overwrites the oldest record. The `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter specifies the minimum age in days of a record before it can be reused.

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how to manage the handling of circular reuse records

Noncircular Reuse Records Noncircular reuse records contain critical information that does not change often and cannot be overwritten. Some examples of information in noncircular reuse records include datafiles, online redo logs, and redo threads.

Recovery Without a Recovery catalog

To make it easier to restore and recover the database without using a recovery catalog, Oracle Corporation recommends that you:

- Enable the **control file autobackup** feature, which causes RMAN to automatically back up the control file, and also enables RMAN to restore the control file autobackup without access to a repository

- **Keep a record of your DBID**, which you may need to recover your database in the event that you lose your control file
- Use a minimum of two multiplexed or mirrored control files **on separate disks**
- Keep all Recovery Manager backup logs.

If you lose the current control files, then you can restore a control file autobackup even if you do not use a recovery catalog.

> **See Also:** "Control File and Server Parameter File Autobackups" on page 2-38 to learn about disaster recovery using control file autobackups

Media Management

Oracle Corporation's Media Management Layer (MML) API lets third-party vendors build a **media manager**, software that works with RMAN and the vendor's hardware to allow backups to sequential media devices such as tape drives. The media manager handles loading, unloading and labeling of sequential media such as tapes. You must install media manager software to use RMAN with sequential media devices.

When backing up or restoring, the RMAN client connects to the target instance and directs the instance to send requests to its media manager. No direct communication occurs between the RMAN client and media manager.

Performing Backup and Restore with a Media Manager

Before performing backup or restore to a media manager, you must allocate one or more channels to handle the commuication with the media manager. You can also configure default channels for use with the media manager, which will be applied for all backup and recovery tasks that use the media manager where you do not explicitly allocate channels. For a conceptual overview of channels, see "About RMAN Channels" on page 2-2. Configuring or allocating channels for backups is discussed further in "Configuring and Allocating Channels for Use in Backups" on page 7-2.

For example, this sequence of commands would configure channels for the media manager and back up the database to the media manager:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt PARMS 'ENV=(NSR_SERVER=bksvr1)';
```

```
RMAN> BACKUP DATABASE;
```

When RMAN executes the BACKUP DATABASE command, it sends the backup request to the database server session performing the backup. The database server session identifies the output channel as a media management device and makes a request to the media manager to write the output.

RMAN does not issue specific commands to load, label, or unload tapes. When backing up, RMAN gives the media manager a stream of bytes and associates a unique name with that stream. When RMAN needs to restore the backup, it asks the media manager to retrieve the byte stream. All details of how and where that stream is stored are handled entirely by the media manager.

The media manager labels and keeps track of the tape and names of files on each tape, and automatically loads and unloads tapes, or signals an operator to do so.

Some media managers support **proxy copy** functionality, in which they handle the entire data movement between datafiles and the backup devices. Such products may use technologies such as high-speed connections between storage and media subsystems to reduce load on the primary database server. RMAN provides a list of files requiring backup or restore to the media manager, which in turn makes all decisions regarding how and when to move the data.

Backup Solutions Program

The Oracle Backup Solutions Program (BSP), part of the Oracle Partner Program, is a group of leading media manager vendors whose products are compliant with Oracle Corporation's MML specification. Several products may be available for your platform from media management vendors. For more information, you can contact your Oracle representative for a list of available products, contact individual vendors to ask them if they participate, or access the Backup Solutions Program Web site at:

```
http://otn.oracle.com/deploy/availability
```

Note that Oracle Corporation does not certify media manager vendors for compatibility with RMAN. Questions about availability, version compatibility, and functionality can only be answered by the media manager vendor, not Oracle Corporation.

2

RMAN Backups Concepts

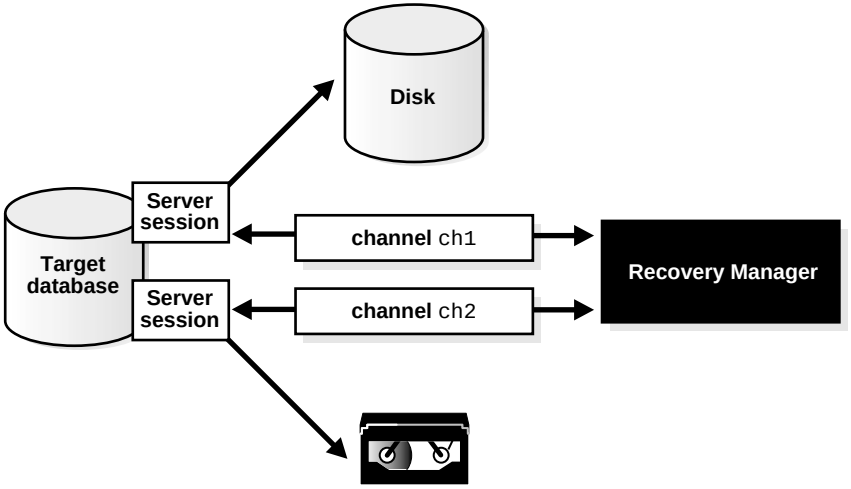
This chapter describes the basic concepts involved in backing up the database with the Recovery Manager (RMAN) utility.

This chapter contains these topics:

About RMAN Channels

An RMAN **channel** represents one stream of data to a device type, and corresponds to one server session. Most backup and recovery commands in RMAN are executed by server sessions. As illustrated in Figure 2–1, each channel establishes a connection from the RMAN client to a target or auxiliary database instance by starting a server session on the instance. The server session performs the backup, restore, and recovery.

Figure 2–1 Channel Allocation



You can use the `CONFIGURE CHANNEL` command to configure channels for use with disk or tape in all RMAN sessions using **automatic channel allocation**, or allocate channels manually within a RUN block. RMAN comes preconfigured with one `DISK` channel that you can use for backups to disk.

When you run a command that requires a channel without allocating a channel explicitly, then RMAN automatically allocates the channels with the options specified in the `CONFIGURE` command. For the `BACKUP` command, RMAN allocates only a single type of channel, such as `DISK`. For the `RESTORE` command and maintenance commands (for example, `DELETE`), RMAN allocates all necessary channels for the device types required to execute the command.

To specify the device type to use for an operation explicitly, use the ALLOCATE CHANNEL command, which must be used within a RUN block, or ALLOCATE CHANNEL FOR MANTAINANCE, which must be executed at the RMAN prompt.

In a Real Application Clusters configuration, there are special considerations regarding channel allocation and backups. See *Oracle Real Application Clusters Administrator's Guide* for more details.

How and when the ALLOCATE CHANNEL or CONFIGURE CHANNEL commands cause the media manager to allocate resources is vendor-specific. Some media managers allocate resources when you issue the command; others do not allocate resources until you open a file for reading or writing.

> **See Also:** *Oracle Database Recovery Manager Reference* for ALLOCATE CHANNEL syntax and *Oracle Database Recovery Manager Reference* on ALLOCATE CHANNEL FOR MAINTENANCE

Automatic and Manual Channel Allocation

You can use the automatic channel allocation feature to configure a set of persistent, automatic channels for use in all RMAN sessions. You can use the manual channel allocation feature to specify channels for commands used within a RUN block.

RMAN allocates automatic channels according to the settings in these commands:

- CONFIGURE DEVICE TYPE ... PARALLELISM
- CONFIGURE DEFAULT DEVICE TYPE
- CONFIGURE CHANNEL

For example, you can issue the following commands at the RMAN prompt:

```
# since you do not manually allocate channels, RMAN uses preconfigured channels
BACKUP DATAFILE 3;
RESTORE TABLESPACE users;
```

When you run a command that requires channels, and no channels have been allocated using the ALLOCATE command, RMAN automatically allocates channels according to values set with the CONFIGURE command in the following cases:

- You use commands such as BACKUP, RESTORE, or DELETE outside of a RUN block.
- You use commands within a RUN block but do not allocate any channels within the RUN block.

You can override automatic channel allocation settings by manually allocating channels within a RUN block. Manual channels always override automatic channels. For example, you override automatic channel allocation when you issue a command as follows:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt;
  BACKUP DATABASE PLUS ARCHIVELOG;
}
```

RMAN optimizes automatic channel allocation by leaving automatic channels allocated so long as each new command requires exactly the same channel configuration as the previous command. For example, RMAN can use the same preallocated channels for the following series of commands:

```
BACKUP DATAFILE 1;
BACKUP CURRENT CONTROLFILE;
BACKUP ARCHIVELOG ALL;
```

If you issue a command such as ALLOCATE or CONFIGURE, then RMAN automatically releases the preallocated channels.

> **See Also:** "Configuring Automatic Channels" on page 6-12 to learn how to configure automatic channels

Automatic Channel Device Configuration and Parallelism

The CONFIGURE DEVICE TYPE . . . PARALLELISM command specifies the number of automatic channels to allocate for a specified device type. For example, if you configure parallelism to 3 for a device type, then RMAN allocates three channels for the device type when using automatic channels.

You can change a parallelism setting by issuing another CONFIGURE DEVICE TYPE . . . PARALLELISM command. This example configures PARALLELISM 2 and then changes it to 3:

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 2;
CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
```

The parallelism setting defines the number of channels for a device that RMAN allocates in parallel. It does not have to correspond to the actual number of channels configured for the device. For example, you can manually configure four different sbt channels and set PARALLELISM for sbt to 2, 1, or 10.

You can view the default setting for parallelism by running the SHOW DEVICE TYPE command. For example:

```
RMAN> SHOW DEVICE TYPE;
RMAN configuration parameters are:
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; #default
```

As always when the SHOW command is used to view the value of a parameter, RMAN includes a "#default" comment at the end of the line if the RMAN default value has not been overridden.

The following example configures the default device to sbt and then displays the resulting configuration using the SHOW DEVICE TYPE command:

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
new RMAN configuration parameters:
CONFIGURE DEFAULT DEVICE TYPE TO 'sbt';
new RMAN configuration parameters are successfully stored

RMAN> SHOW DEVICE TYPE;
RMAN configuration parameters are:
CONFIGURE DEVICE TYPE SBT PARALLELISM 1; # default
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; #
default
```

See Also: "Configuring a Generic Automatic Channel for a Device Type" on page 6-13

Automatic Channel Default Device Types

Run the CONFIGURE DEFAULT DEVICE TYPE command to specify a default device type for automatic channels. For example, you may make backups to tape most of the time and only occasionally make a backup to disk. In this case, configure channels for disk and tape devices, but make sbt the default device type:

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 1; # configure device disk
CONFIGURE DEVICE TYPE sbt PARALLELISM 2; # configure device sbt
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

Now, RMAN will, by default, use sbt channels for backups. For example, if you run the following command:

```
BACKUP TABLESPACE users;
```

RMAN only allocates channels of type sbt during the backup because sbt is the default device.

You can override the default device for backups by specifying a different device on the command. For example:

```
BACKUP DEVICE TYPE sbt DATABASE;
```

If the default device type is DISK, then the preceding command overrides this default and uses the sbt channel configuration. Note that this command fails unless you have configured the sbt device or configured sbt channels.

When restoring files, RMAN allocates all automatic channels according to the settings configured for each device type. The default device type configuration is irrelevant. For example, if you configure PARALLELISM to 3 for the default sbt device and PARALLELISM to 2 for DISK, then RMAN automatically allocates three sbt channels and two DISK channels during the restore.

Automatic Channel Naming Conventions

RMAN uses the following convention for channel naming: ORA_*devicetype*_*n*, where *devicetype* refers to the user's device type (such as DISK or sbt_tape) and *n* refers to the channel number.

> **Note:** The sbt and sbt_tape device types are synonymous, but RMAN output always displays sbt_tape whether the input is sbt or sbt_tape.

For example, RMAN names the first DISK channel ORA_DISK_1, the second ORA_DISK_2, and so forth. RMAN names the first sbt channel ORA_SBT_TAPE_1, the second ORA_SBT_TAPE_2, and so forth. When you parallelize channels, RMAN always allocates channels in numerical order, starting with 1 and ending with the parallelism setting (CONFIGURE DEVICE TYPE . . . PARALLELISM *n*), as in this example:

```
ORA_SBT_TAPE_1
ORA_SBT_TAPE_2
ORA_SBT_TAPE_3
```

Automatic channel allocation also applies to maintenance commands. If RMAN allocates an automatic maintenance channel, then it uses the same naming convention as any other automatically allocated channel. If you manually allocate a maintenance channel using ALLOCATE CHANNEL FOR MAINTENANCE, then RMAN uses the following convention for channel naming: ORA_MAINT_*devicetype*_*n*,

where *devicetype* refers to the user's device type (for example, DISK or sbt) and *n* refers to the channel number. For example, RMAN uses these names for two manually allocated disk channels:

```
ORA_MAINT_DISK_1
ORA_MAINT_DISK_2
```

Note that if you run the CONFIGURE DEVICE TYPE command to configure a device type and do not run CONFIGURE CHANNEL for this device type, then RMAN allocates all channels without other channel control options. For example, assume that you configure the sbt device and run a backup as follows:

```
CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
BACKUP DEVICE TYPE sbt DATABASE;
```

In effect, RMAN does the following:

```
RUN
{
  ALLOCATE CHANNEL ORA_SBT_TAPE_1 DEVICE TYPE sbt;
  BACKUP DATABASE;
}
```

Channel names beginning with the ORA_ prefix are reserved by RMAN for its own use. You cannot manually allocate a channel with a name that begins with ORA_.

Automatic Channel Generic Configurations

The CONFIGURE CHANNEL DEVICE TYPE command configures generic settings that are used for all automatic channels of the specified device type. In other words, the command creates a template of settings that RMAN uses for all channels allocated on the device. For example, you can configure disk and tape channels as follows:

```
CONFIGURE CHANNEL DEVICE TYPE sbt PARMS='ENV=(NSR_SERVER=bksvr1)';
CONFIGURE CHANNEL DEVICE TYPE DISK RATE 5M FORMAT="?/oradata/%U";
```

Because you do not specify channel numbers for these channels, the channel settings are generic to all automatic channels of the specified type. The configuration acts as a template. For example, if you set PARALLELISM for DISK to 10, and the default device type is DISK, then RMAN allocates ten disk channels using the settings in the CONFIGURE CHANNEL DEVICE TYPE DISK command.

See Also: "Configuring a Generic Automatic Channel for a Device Type" on page 6-13

Automatic Channel-Specific Configurations

You can also configure parameters that apply to a specific automatic channel. If you are using a media manager that requires different settings on each channel, then you may find it useful to configure individual channels.

You can mix a CONFIGURE CHANNEL command that creates a generic configuration with a CONFIGURE CHANNEL command that creates a specific configuration. A generic automatic channel creates a configuration that can be used for any channel that is not explicitly configured.

For example, assume that you run these commands:

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE = 2M;
CONFIGURE CHANNEL 3 DEVICE TYPE DISK MAXPIECESIZE = 900K;
```

In this scenario, RMAN allocates ORA_DISK_1 and ORA_DISK_2 with option MAXPIECESIZE = 2M, using the settings for the DISK channel with no number. RMAN allocates ORA_DISK_3 with MAXPIECESIZE = 900K because this channel was manually assigned a channel number. RMAN always allocates the number of channels specified in the parallelism parameter.

> **See Also:** "Configuring a Generic Automatic Channel for a Device Type" on page 6-13

Clearing Automatic Channel Settings

You can specify the CLEAR option for any CONFIGURE command. The CLEAR option returns the specified configuration to its default value. Assume you run these commands:

```
CONFIGURE DEVICE TYPE DISK CLEAR;     # returns DISK to default PARALLELISM 1
                                      # and backup type to BACKUPSET
CONFIGURE DEFAULT DEVICE TYPE CLEAR;    # returns to default device type of DISK
CONFIGURE CHANNEL DEVICE TYPE sbt CLEAR; # removes all options for sbt channel
CONFIGURE CHANNEL 3 DEVICE TYPE DISK CLEAR; # removes configurations for 3rd ch.
```

Each CONFIGURE...CLEAR command removes the user-entered settings and returns the configuration to its default value.

The only way to find out the default setting for parameters set through CONFIGURE is to use CONFIGURE... CLEAR to un-set the parameter, so that it takes on the default value, and then run SHOW ALL to view all parameters. For any parameter for which

the value is currently set to RMAN'S default, RMAN includes a "#default" comment at the end of that line of the output from SHOW ALL.

> **See Also:** *Oracle Database Recovery Manager Reference* for the default settings for each CONFIGURE command

Determining Channel Parallelism to Match Hardware Devices

RMAN can perform the I/O required for many commands in parallel, to make optimal use of your hardware resources. To perform I/O in parallel, however, the I/O must be associated with a single RMAN command, not a series of commands. For example, if backing up three datafiles, issue the command

```
BACKUP DATAFILE 5,6,7;
```

rather than issuing the commands

```
BACKUP DATAFILE 5;
BACKUP DATAFILE 6;
BACKUP DATAFILE 7;
```

When all three datafiles are backed up in one command, RMAN recognizes the opportunity for parallelism and can use multiple channels to do the I/O in parallel. When three separate commands are used, RMAN can only perform the backups one at a time, regardless of available channels and I/O devices.

The number of channels available (whether allocated in a RUN block or configured in advance) for use with a device at the moment that you run a command determines whether RMAN will read from or write to that device in parallel while carrying out the command. Failing to allocate the right number of channels adversely affects RMAN performance during I/O operations.

As a rule, the number of channels used in carrying out an individual RMAN command should match the number of physical devices accessed in carrying out that command. If manually allocating channels for a command, allocate one for each device; if configuring automatic channels, configure the PARALLELISM setting appropriately.

When backing up to tape, you should allocate one channel for each tape drive. When backing up to disk, allocate one channel for each physical disk, unless you can optimize the backup for your disk topography by using multiple disk channels. Each manually allocated channel uses a separate connection to the target or auxiliary database.

The following script creates three backups sequentially: three separate BACKUP commands are used to back up one file each. Only one channel is active at any one time because only one file is being backed up in each command.

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt;
  ALLOCATE CHANNEL c2 DEVICE TYPE sbt;
  ALLOCATE CHANNEL c3 DEVICE TYPE sbt;
  BACKUP DATAFILE 5;
  BACKUP DATAFILE 6;
  BACKUP DATAFILE 7;
}
```

The following statement uses **parallelization** on the same example: one RMAN BACKUP command backs up three datafiles, with all three channels in use. The three channels are **concurrently active**—each server session copies one of the datafiles to a separate tape drive.

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt;
  ALLOCATE CHANNEL c2 DEVICE TYPE sbt;
  ALLOCATE CHANNEL c3 DEVICE TYPE sbt;
  BACKUP DATAFILE 5,6,7;
}
```

> **See Also:** *Oracle Real Application Clusters Administrator's Guide* for information about parallelization in a Real Application Clusters (RAC) configuration. In a RAC configuration you may want to specify different CONNECT strings for each channel, to connect to different instances of the target database and distribute work across the cluster. You may also want to use more than one channel for each device in a RAC configuration.

Channel Control Options for Manual and Automatic Channels

Whether you allocate channels manually or automatically, you can use channel control commands and options to do the following:

- Control the operating system resources RMAN uses when performing RMAN operations
- Affect the degree of parallelism for a backup or restore command

- Set limits on I/O bandwidth consumption in kilobytes, megabytes, or gigabytes (ALLOCATE CHANNEL . . . RATE, CONFIGURE CHANNEL . . . RATE)
- Set limits on the size of backup pieces (the MAXPIECESIZE parameter specified on the CONFIGURE CHANNEL and ALLOCATE CHANNEL commands)
- Set limits on the size of backup sets (the MAXSETSIZE parameter specified on the BACKUP and CONFIGURE commands)
- Send vendor-specific commands to the media manager (SEND)
- Specify vendor-specific parameters for the media manager (ALLOCATE CHANNEL . . . PARMS, CONFIGURE CHANNEL . . . PARMS)
- Specify which instance performs the operation (ALLOCATE CHANNEL . . . CONNECT, CONFIGURE CHANNEL . . . CONNECT)

In releases 8.1.5 and later of the Oracle database, the ALLOCATE CHANNEL command causes RMAN to contact the media manager whenever the type specified is other than DISK. In earlier releases, the ALLOCATE CHANNEL command does not cause RMAN to contact the media manager; RMAN does not call the media manager until a BACKUP, RESTORE, or RECOVER command is issued.

Note: When you specify DEVICE TYPE DISK with any version of RMAN, RMAN does not allocate operating system resources other than for the creation of the server session and does not call the media manager.

Because RMAN has one preconfigured automatic DISK channel, you do not have to manually allocate a maintenance channel when running CHANGE, CROSSCHECK, or DELETE against a disk file (that is, an ARCHIVELOG, DATAFILECOPY, or CONTROLFILECOPY).

A maintenance channel is useful only for a maintenance task; you cannot use it as an input or output channel for a backup or restore.

> **See Also:** *Oracle Database Recovery Manager Reference* for ALLOCATE CHANNEL syntax, and *Oracle Database Recovery Manager Reference* for CONFIGURE syntax

Channel Failover

A BACKUP command is decomposed into multiple independent backup steps by RMAN. Each independent step can be executed on any channel allocatedfor the

type of device used in the command. If you have multiple channels allocated, and one channel fails or encounters a problem during a backup step, then RMAN attempts to complete the work on another channel. Typically, such retriable errors can occur when a media manager encounters problems with one of several tape drives, or when an instance fails in a RAC environment.

RMAN reports a message in V$RMAN_OUTPUT and in the output to the interactive session or log file when it encounters such problems, as in the following example (refer to bold text):

```
channel ORA_SBT_TAPE_1: backup set failed, re-triable on other channel
ORA-19506: failed to create sequential file, name="/bkup/63d3c3og_1_1", parms=""
ORA-27028: skgfqcre: sbtbackup returned error
ORA-19511: Error received from media manager layer, error text: failed to open
           file /bkup/63d3c3og_1_1 for backup, errno = 2
channel ORA_SBT_TAPE_2: finished piece 1 at 06-SEP-01 piece handle=5ld3blun_1_1
                        comment=API Version 2.0,MMS Version 3.2.0.0
channel ORA_SBT_TAPE_2: backup set complete, elapsed time: 00:00:04
retrying ORA_SBT_TAPE_1 failed backup step on ORA_SBT_TAPE_2
channel ORA_SBT_TAPE_2: starting full datafile backupset
channel ORA_SBT_TAPE_2: specifying datafile(s) in backupset input datafile
                        fno=00004 name=/oracle/dbs/tbs_12.f input datafile
                        fno=00017 name=/oracle/dbs/tbs_14.f
channel ORA_SBT_TAPE_2: starting piece 1 at 06-SEP-01 piece handle=5ld3buds_1_1
                        comment=API Version 2.0,MMS Version 3.2.0.0
channel ORA_SBT_TAPE_2: backup set complete, elapsed time: 00:00:06
```

Note that if RMAN is executing a script, then the next command in the script will not be executed if there were any errors in the previous command.

> **See Also:** "Interpreting RMAN Message Output" on page 15-2 to learn more about RMAN message and error reporting

About RMAN Backups

When you execute the BACKUP command in RMAN, you create one or more backup sets or image copies. By default, RMAN creates backup sets regardless of whether the destination is disk or a media manager.

About Image Copies

An **image copy** is an exact copy of a single datafile, archived redo log file, or control file. Image copies are not stored in an RMAN-specific format. They are identical to the results of copying a file with operating system commands. RMAN can use

image copies during RMAN restore and recover operations, and you can also use image copies with non-RMAN restore and recovery techniques.

To create image copies and have them recorded in the RMAN repository, run the RMAN BACKUP AS COPY command (or, alternatively, configure the default backup type for disk as image copies using CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY before performing a backup). A database server session is used to create the copy, and the server session also performs actions such as validating the blocks in the file and recording the image copy in the RMAN repository.

You can also use an operating system command such as the UNIX dd command to create image copies, though these will not be validated, nor are they recorded in the RMAN repository. You can use the CATALOG command to add image copies created with native operating system tools in the RMAN repository.

Using RMAN-Created Image Copies

If you run a RESTORE command, then by default RMAN restores a datafile or control file to its original location by copying an image copy backup to that location. Image copies are chosen over backup sets because of the extra overhead of reading through an entire backup set in search of files to be restored.

However, if you need to restore and recover a current datafile, and if you have an image copy of the datafile available on disk, then you do not actually need to have RMAN copy the image copy back to its old location. You can instead have the database use the image copy in place, as a replacement for the datafile to be restored. The SWITCH command updates the RMAN repository indicate that the image copy should now be treated as the current datafile. Issuing the SWITCH command in this case is equivalent to issuing the SQL statement ALTER DATABASE RENAME FILE. You can then perform recovery on the copy.

User-Managed Image Copies

RMAN can use image copies created by mechanisms outside of RMAN, such as native operating system file copy commands or third-party utilities that leave image copies of files on disk. These copies are known as **user-managed copies** or **operating system copies**.

The RMAN CATALOG command causes RMAN to inspect an existing image copy and enter its metadata into the RMAN repository. Once cataloged, these files can be used like any other backup with the RESTORE or SWITCH commands.

Some sites store their datafiles on mirrored disk volumes, which permit the creation of image copies by **breaking a mirror**. After you have broken the mirror, you can

notify RMAN of the existence of a new user-managed copy, thus making it a candidate for a backup operation. You must notify RMAN when the copy is no longer available, by using the CHANGE . . . UNCATALOG command. In this example, before **resilvering the mirror** (not including other copies of the broken mirror), you must use a CHANGE . . . UNCATALOG command to update the recovery catalog and indicate that this copy is no longer available.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to catalog datafile and archived log image copies
- "Making Split Mirror Backups with RMAN" on page 7-5
- *Oracle Database Recovery Manager Reference* for CHANGE syntax

About Proxy Copies

During a **proxy copy**, RMAN turns over control of the data transfer to a media manager that supports this feature. Proxy copy can only be used with media managers that support it, not with disk. The PROXY option of the BACKUP command specifies that a backup should be a proxy copy.

For each file that you attempt to back up with the BACKUP PROXY command, RMAN queries the media manager to determine whether it can perform a proxy copy. If the media manager cannot proxy copy the file, then RMAN backs the file up as if the PROXY option had not been used. (Use the PROXY ONLY option to force RMAN to fail if a proxy copy cannot be performed.)

Proxy copy can be used with datafiles or archived redo logs, as shown in these examples:

```
BACKUP DEVICE TYPE sbt PROXY DATAFILE 3;
BACKUP DEVICE TYPE sbt PROXY ONLY DATABASE;
BACKUP DEVICE TYPE sbt PROXY ONLY ARCHIVELOG ALL;
```

The examples assume that sbt channels have been configured with the appropriate parameters.

Note that control files are never backed up with proxy copy. If the PROXY option is specified on an operation backing up a control file, it is silently ignored for the purposes of backing up the control file.

See Also:

- *Oracle Database Reference* for more information about V$PROXY_DATAFILE and V$PROXY_ARCHIVEDLOG
- *Oracle Database Recovery Manager Reference* for the BACKUP command and the PROXY option

Storage of Backups on Disk and Tape

RMAN can create backups on disk or a third-party media device such as a tape drive. If you specify DEVICE TYPE DISK, then your backups are created on disk, in the file name space of the target instance that is creating the backup. You can make a backup on any device that can store a datafile.

To create backups on non-disk media, such as tape, you must use third-party media management software, and allocate channels with device types, such as SBT, that are supported by that software.

Backups of Archived Logs

There are several features of RMAN backups specific to backups of archived redo logs.

Deletion of Archived Logs After Backups

RMAN can delete one or all copies of archived logs from disk after backing them up to backup sets. If you specify the DELETE INPUT option, then RMAN backs up exactly one copy of each specified log sequence number and thread from an archive destination to tape, and then deletes the specific file it backed up while leaving the other copies on disk. If you specify the DELETE ALL INPUT option, then RMAN backs up exactly one copy of each specified log sequence number and thread, and then deletes that log from all archive destinations. Note that there are special considerations related to deletion of archived redo logs in standby database configurations. See *Oracle Data Guard Concepts and Administration* for details.

Backup Failover for Archived Redo Logs

RMAN's **archived redo log failover** allows RMAN to complete a backup even when some archived log destinations are missing logs or have logs with corrupt blocks. If at least one log corresponding to a given log sequence and thread is available in any of the archiving destinations, then RMAN tries to back it up. If RMAN finds a corrupt block in a log file during backup, it searches other destinations for a copy of that log without corrupt blocks.

By default, RMAN only backs up one copy of each distinct log sequence number. For example, assume that you archive logs 121 through 124 to two archiving destinations: `/arch1` and `/arch2`. The control file contains archived log records as follows:

Sequence	Filename in /arch1	Filename in /arch2
121	`/arch1/archive1_121.arc`	`/arch2/archive1_121.arc`
122	`/arch1/archive1_122.arc`	`/arch2/archive1_122.arc`
123	`/arch1/archive1_123.arc`	`/arch2/archive1_123.arc`
124	`/arch1/archive1_124.arc`	`/arch2/archive1_124.arc`

However, unknown to RMAN, a user deletes logs 122 and 124 from the `/arch1` directory. Then, you run the following backup:

```
BACKUP ARCHIVELOG FROM SEQUENCE 121 UNTIL SEQUENCE 125;
```

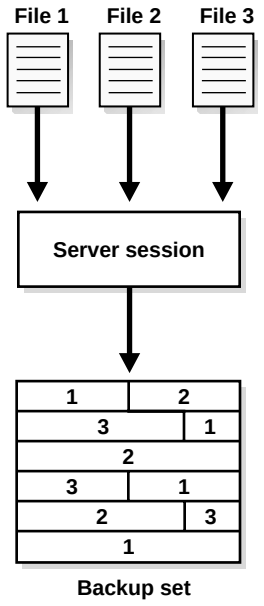
With failover, RMAN completes the backup, using logs 122 and 124 in `/arch2`.

Multiplexed Backup Sets

When creating backup sets, you can **multiplex** files. In this case, RMAN simultaneously reads multiple files from disk and and then writes their blocks into the same backup set. (Image copies, by contrast, are never multiplexed.) For example, RMAN can read from two datafiles simultaneously, and then combine the blocks from these datafiles into a single backup piece.

As Figure 2–2 illustrates, RMAN can back up three datafiles into a backup set that contains only one backup piece. This backup piece contains the intermingled data blocks of the three input files.

Figure 2–2 Datafile Multiplexing



Algorithm for Multiplexed Backups

RMAN multiplexing is determined by the following algorithm:

1. The **number of files in each backup set** is determined by computing the lesser of the following values:
   - The default number of files in a backup set (16 for archived logs, and 4 for datafiles)
   - The number of files read by each channel.
2. The **level of multiplexing** is determined by the lesser of the following values:
   - The number of files in the backup set (as calculated by the preceding step)
   - The default number of files that RMAN reads simultaneously on a single channel (8 files for each channel)

Assume that you are backing up twelve datafiles with one RMAN channel. The number of files in each backup set is 4. To determine the level of multiplexing, compare this value to 8 and take the lesser, which is 4. Because the level of

multiplexing is 4, the channel includes blocks from four separate datafiles into each backup set.

See Also:

- "I/O Buffer Allocation" on page 14-2 to learn how multiplexing affects allocation of disk buffers during backups
- *Oracle Database Recovery Manager Reference* for BACKUP syntax

Multiplexing by the Media Manager

Media manager multiplexing occurs when the media manager writes the concurrent output from multiple RMAN channels to a single sequential device, such as a tape drive.

Caution: Although media manager multiplexing can sometimes provide a performance benefit during backup, it can have a negative impact on restore performance. Oracle Corporation recommends using RMAN multiplexing instead of using multiplexing by the media manager.

Manual Parallelization of Backups

When you configure PARALLELISM to greater than 1 or manually allocate multiple channels, RMAN writes multiple backups sets or image copies in parallel. The channels divide the work of backing up the specified files.

Note: You cannot stripe a single backup set across multiple channels.

By default, RMAN determines which channels should back up which database files. You can use the CHANNEL option of the BACKUP command to manually assign a channel to back up specified files. This example shows a parallelized backup to the default disk location that uses the default automatic DISK channels:

```
BACKUP
  (DATAFILE 1,2,3
   CHANNEL ORA_DISK_1)
  (DATAFILECOPY '/tmp/system01.dbf', '/tmp/tools01.dbf'
   CHANNEL ORA_DISK_2)
  (ARCHIVELOG FROM SEQUENCE 100 UNTIL SEQUENCE 102 THREAD 1
```

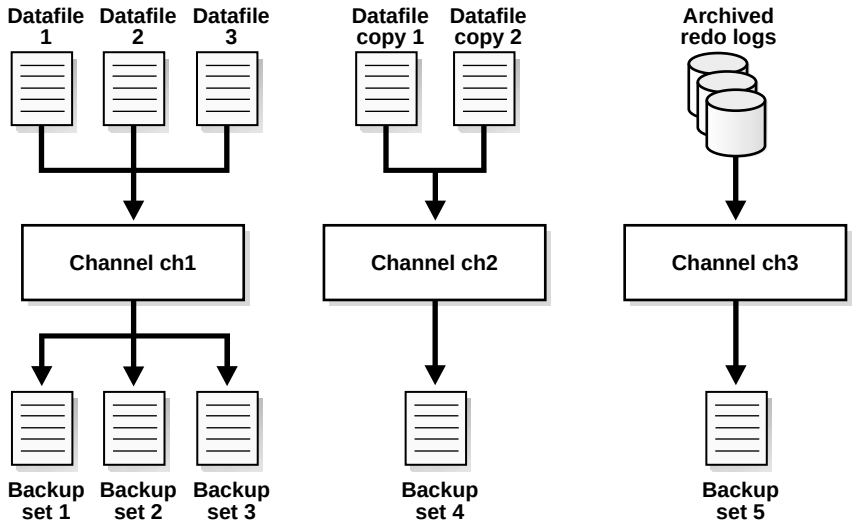
```
   CHANNEL ORA_DISK_3);
```

You can also manually allocate channels as in the following example:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS="ENV=(BACKUP_SERVER=tape_server1)";
  ALLOCATE CHANNEL c2 DEVICE TYPE sbt PARMS="ENV=(BACKUP_SERVER=tape_server2)";
  ALLOCATE CHANNEL c3 DEVICE TYPE sbt PARMS="ENV=(BACKUP_SERVER=tape_server3)";
  BACKUP
   (DATAFILE 1,2,3
    CHANNEL c1)
   (DATAFILECOPY '/tmp/system01.dbf', '/tmp/tools01.dbf'
     CHANNEL c2)
   (ARCHIVELOG FROM SEQUENCE 100 UNTIL SEQUENCE 102 THREAD 1
     CHANNEL c3);
}
```

Figure 2–3 shows an example of parallelization in which channel ch1 backs up datafiles, channel ch2 backs up datafile copies, and channel ch3 backs up logs.

Figure 2–3 Parallelization of Backups



See Also:

- "Determining Channel Parallelism to Match Hardware Devices" on page 2-9 for an overview of how allocated channels affect parallelization
- "Determining How Channels Distribute a Backup Workload: Example" on page 7-20 to learn how to parallelize backups
- *Oracle Database Recovery Manager Reference* for reference material on the CHANNEL parameter of the BACKUP command

Multiple Copies of RMAN Backups

In RMAN, there are two ways to make multiple, identical copies of backups:

- Duplex your backups within the BACKUP AS BACKUPSET command, in which case RMAN creates more than one copy of each backup set
- Back up your files as backup sets or image copies, and then back up the backup sets or image copies using the RMAN BACKUP BACKUPSET or BACKUP COPY commands.

Duplexed Backup Sets

When backing up datafiles, archived redo log files, server parameter files and control files into backup pieces, RMAN can **duplex** the backup set, producing up to four identical copies of each backup piece in the backup set on different backup destinations with one BACKUP command. (Note that duplexing is not supported for backup operations that produce image copies.)

There are three ways to specify duplexing of backup sets when using the BACKUP command:

- **Specify a default level of duplexing with CONFIGURE... BACKUP COPIES**

  All backup commands that back up data into backup sets will be affected if you use this option, unless you specify different duplexing options for a command using SET BACKUP COPIES or provide a COPIES option for the BACKUP command.

- **Use SET BACKUP COPIES in a RUN block**

  All commands in the RUN block will be affected, overriding any CONFIGURE... BACKUP COPIES setting, except those where you provide a COPIES option as part of the BACKUP command.

- **Provide a COPIES option to the BACKUP command**

  For this specific BACKUP command, files will be duplexed to produce the number of copies you specify.

The FORMAT option of the BACKUP command specifies the destinations to be used when performing duplexed backups. You can specify up to 4 values for the FORMAT option. RMAN uses the second, third, and fourth values only when BACKUP COPIES, SET BACKUP COPIES, or CONFIGURE . . . BACKUP COPIES is specified. The following example creates 3 copies of the backup of datafile 7:

```
BACKUP DEVICE TYPE DISK COPIES 3 DATAFILE 7 FORMAT
'/tmp/%U','?/oradata/%U','?/%U';
```

RMAN places the first copy of each backup piece in /tmp, the second in ?/oradata, and the third in the Oracle home. Note that RMAN does not produce 3 backup sets, each with a different unique backup set key. Rather, RMAN produces one backup set with a unique key, and generates 3 identical copies of each backup piece in the set, as shown in this sample LIST output:

```
List of Backup Sets
===================

BS Key  Type LV Size
------- ---- -- ----------
1       Full    64K
  List of Datafiles in backup set 1
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  7       Full 98410      08-FEB-03 /oracle/oradata/trgt/tools01.dbf

  Backup Set Copy #1 of backup set 1
  Device Type Elapsed Time Completion Time Tag
  ----------- ------------ --------------- ---
  DISK        00:00:01     08-FEB-03       TAG20030208T152314

    List of Backup Pieces for backup set 1 Copy #1
    BP Key  Pc# Status      Piece Name
    ------- --- ----------- ----------
    1       1   AVAILABLE   /tmp/01dg9tb2_1_1

  Backup Set Copy #2 of backup set 1
  Device Type Elapsed Time Completion Time Tag
  ----------- ------------ --------------- ---
  DISK        00:00:01     08-FEB-03       TAG20030208T152314
```

```
  List of Backup Pieces for backup set 1 Copy #2
  BP Key  Pc# Status      Piece Name
  ------- --- ----------- ----------
  2       1   AVAILABLE   /oracle/oradata/01dg9tb2_1_2

Backup Set Copy #3 of backup set 1
Device Type Elapsed Time Completion Time Tag
---------- ------------ --------------- ---
DISK        00:00:01     08-FEB-03       TAG20030208T152314

  List of Backup Pieces for backup set 1 Copy #3
  BP Key  Pc# Status      Piece Name
  ------- --- ----------- ----------
  3       1   AVAILABLE   /oracle/01dg9tb2_1_3
```

When choosing which FORMAT value to use for each backup piece, RMAN uses the first format value for copy number 1, the second format value for copy number 2, and so forth. If the number of format values exceeds the number of copies, then the extra formats are not used. If the number of format values is less than the number of copies, then RMAN reuses the format values, starting with the first one.

See Also:

- "Duplexing Backup Sets" on page 7-3 to learn how to duplex backups
- *Oracle Database Recovery Manager Reference* for CONFIGURE syntax
- *Oracle Database Recovery Manager Reference* for SET syntax

Backups of Backup Sets

The RMAN BACKUP BACKUPSET command backs up previously created backup sets. Only backup sets that were created on device type DISK can be backed up, and they can be backed up to any available device type.

Note: RMAN issues an error if you attempt to run BACKUP AS COPY BACKUPSET.

The BACKUP BACKUPSET command uses the default disk channel to copy backup sets from disk to disk. To back up from disk to tape, you must either configure or manually allocate a non-disk channel.

Uses for Backups of Backup Sets

The BACKUP BACKUPSET command is a useful way to spread backups among multiple media. For example, you can execute the following BACKUP command weekly as part of the production backup schedule:

```
# makes backup sets on disk
BACKUP DEVICE TYPE DISK AS BACKUPSET DATABASE PLUS ARCHIVELOG;
BACKUP DEVICE TYPE sbt BACKUPSET ALL; # copies backup sets on disk to tape
```

> **Note:** Backups to sbt that use automatic channels require that you first run the CONFIGURE DEVICE TYPE sbt command.

In this way, you ensure that all your backups exist on both disk and tape. You can also duplex backups of backup sets, as in this example:

```
BACKUP COPIES 2 DEVICE TYPE sbt BACKUPSET ALL;
```

(Again, control file autobackups are never duplexed.)

You can also use BACKUP BACKUPSET to manage backup space allocation. For example, to keep more recent backups on disk and older backups only on tape, you can regularly run the following command:

```
BACKUP DEVICE TYPE sbt BACKUPSET COMPLETED BEFORE 'SYSDATE-7' DELETE INPUT;
```

This command backs up backup sets that were created more than a week ago from disk to tape, and then deletes them from disk. Note that DELETE INPUT here is equivalent to DELETE ALL INPUT;RMAN deletes all existing copies of the backup set. If you duplexed a backup to four locations, then RMAN deletes all four copies of the pieces in the backup set.

Backup Optimization When Backing Up Backup Sets

If **backup optimization** is enabled when you issue the command to back up a backup set, and if the identical backup set has already been backed up to the same device type, then RMAN skips the backup of this backup set. For example, when backup optimization is turned on, the following command backs up to tape only those backup sets not already backed up on device type sbt:

```
BACKUP DEVICE TYPE sbt BACKUPSET ALL;
```

Backup Failover When Backing Up Backup Sets

When backing up backup sets, if RMAN discovers that one copy of a backup set is corrupted or missing, then it searches for other copies of the same backup set, based on the RMAN repository records about the backup set. This behavior is similar to the behavior of RMAN when backing up archived redo logs that exist in multiple archiving destinations.

For example, assume that backup set with key 872 contains three backup pieces, and that `BACKUP COPIES 3` was issued so that three copies of each backup piece were created, each on a different file system. Also assume that some copies have been deleted or corrupted, so that the following table describes the current status of the backup copies:

Backup Piece Number	Copy Number of the Piece	Status of Copy
1	1	Corrupted
1	2	Intact
1	3	Corrupted
2	1	Missing
2	2	Corrupted
2	3	Intact
3	1	Intact
3	2	Corrupted
3	3	Missing

The following command will cause RMAN to perform automatic failover:

```
BACKUP BACKUPSET 872;
```

RMAN copies only the backup pieces listed as "Intact" in the preceding table in its backup set.

Backups of Image Copies

You can use the following commands to back up image copies of database files either as backup sets or as image copies:

- ```
  BACKUP AS COPY
      COPY OF DATABASE;
  ```
  ```

- BACKUP AS BACKUPSET
  COPY OF TABLESPACE;
- BACKUP AS BACKUPSET
  COPY OF DATAFILE;

When using these commands, there must already exist an image copy of every datafile specified in the command. If there are multiple copies of a datafile, the latest one is used. RMAN issues an error if image copies of every datafile in the database or tablespace do not exist.

RMAN Backup Options: Naming, Sizing, and Speed

Recovery Manager provides a number of options to control filenames, sizes of backups and speed during backup.

Filenames for Backup Pieces

You can either let RMAN determine a unique name for backup pieces or use the FORMAT parameter to specify a name. For example, enter:

```
BACKUP TABLESPACE users;
```

RMAN automatically generates unique names for the backup pieces in the default backup location.

The FORMAT parameter provides substitution variables that you can use to generate unique filenames. For example, you can run a command as follows:

```
BACKUP TABLESPACE users FORMAT = '/tmp/users_%u%p%c';
```

As described in "Manual Parallelization of Backups" on page 2-18, you can specify up to four FORMAT values. RMAN uses the second, third, and fourth values only when you run BACKUP COPIES, SET BACKUP COPIES, or CONFIGURE . . . BACKUP COPIES.

Note: If you use a media manager, then check your vendor documentation for restrictions on FORMAT such as the size of the name, the naming conventions, and so forth.

> **See Also:** *Oracle Database Recovery Manager Reference* for descriptions of the FORMAT parameter and the substitution variables

Filenames for Image Copies

FORMAT variables are also used to specify the names of image copies. The default format %U is defined differently for image copies than for backup pieces. RMAN produces image copies of three types of files: datafiles, control files, and archived logs. The following table describes the meaning of %U for each type of file.

Type of File	Meaning of %U
Datafile	data-D-%d_id-%I_TS-%N_FNO-%f_%u
Archived log	arch-D_%d-id-%I_S-%e_T-%h_A-%a_%u
Control file	cf-D_%d-id-%I_%u

When creating image copies (and only image copies), you can also name the output copies with the DB_FILE_NAME_CONVERT option of the BACKUP command. This parameter works identically to the initialization parameter DB_FILE_NAME_CONVERT. Pairs of filename prefixes are provided to change the names of the output files. If a file is not converted by any of the pairs, then RMAN uses the FORMAT specification; if no FORMAT is specified, then RMAN uses the default format %U.

For example, you can run the following command to copy the datafiles whose filename is prefixed with /maindisk/oradata/users so that they are prefixed with /backups/users_ts:

```
BACKUP AS COPY TABLESPACE users
  DB_FILE_NAME_CONVERT=('/maindisk/oradata/users','/backups/users_ts');
```

Tags for RMAN Backups

You can assign a user-specified character string called a **tag** to backup sets and image copies (either copies created by RMAN or copies created by an operating system utility). A tag is a symbolic name for a backup set or file copy, such as weekly_backup. You can specify the tag rather than the filename when executing the RESTORE or CHANGE command. The maximum length of a tag is 30 bytes.

Default RMAN Backup Tag Format

If you do not specify a tag, then RMAN creates a default tag for backups (except for control file autobackups) in the format TAG*YYYYMMDD*T*HHMMSS*, where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour (in 24-hour format), *MM* is the minutes, and *SS* is the seconds. For example, a backup of datafile 1 may receive the tag TAG20030208T133437. The date and time refer to when RMAN started the backup, in the time zone of the instance performing the backup. If multiple backup sets are created by one BACKUP command, then each backup piece is assigned the same default tag.

How Tags Are Applied

When applied to a backup set, a tag applies to a specific copy of the backup set. If you do not duplex a backup set, that is, make multiple identical copies of it, then the backup set has just one tag. For example,

```
BACKUP COPIES 1 DATAFILE 7 TAG foo
```

creates one backup set with tag FOO. Tags are stored in uppercase, regardless of the case used when entering them. However, you can back up this backup set and give this new copy of the backup set the tag BAR. So, the backup set has two identical copies: one tagged FOO and the other tagged BAR.

When applied to image copies, a tag applies to each individual image copy. For example, you run the following command:

```
# Back up as image copies the datafiles of tablespaces users and tools
# all copies get the TAG 'users-tools'
BACKUP AS COPY TAG users-tools TABLESPACE users, tools;
```

You can also copy an image copy with a specific tag, and give the output copy a different tag, as in the following example:

```
# Create new copies of all image copies of the database that have the tag
# 'full_cold_copy', and give the new copies the tag 'new_full_cold_copy'
BACKUP AS COPY
  COPY OF DATABASE
    FROM TAG=full_cold_copy
    TAG=new_full_cold_copy;

# Create backup sets of the image copy of tablespace users that has the tag
# 'monday_users', and of tablespace SYSTEM which has the tag 'monday_system'.
# All these new backup # sets receive the tag 'for_audit'.
BACKUP AS BACKUPSET TAG for_audit
  COPY OF TABLESPACE users FROM TAG monday_users
```

```
TABLESPACE SYSTEM FROM TAG monday_system;
```

Uniqueness of Backup Tags

Tags do not need to be unique, so multiple backup sets or image copies can have the same tag, for example, `weekly_backup`. When you specify that a datafile should be restored from backups that have a specific tag, and more than one backup of the requested file has the desired tag, RMAN restores the most recent backup that has the desired tag (within any other constraints of the restore command, of course, such as a point in time).

Tags can indicate the intended purpose or usage of different classes of backups or copies. For example, datafile copies that are suitable for use in a `SWITCH` can be tagged differently (`for_switch_only`) from file copies that should be used only for `RESTORE` (`for_restore_only`).

Note: If you specify the `FROM` tag option to the `RESTORE` or `SWITCH` command, then RMAN considers *only* backup sets and image copies with a matching tag when choosing which backup to use

---

See Also: *Oracle Database Recovery Manager Reference* for `SWITCH` syntax, and *Oracle Database Recovery Manager Reference* for `RESTORE` syntax

Size of Backup Pieces

RMAN will, by default, put the entire contents of a backup set into one backup piece, regardless of the size of the backup set. If you are backing up to a file system or media manager that has a limit on the maximum file size that can be created, then you may need to restrict the size of backup pieces that RMAN will create.To restrict the size of each backup piece, specify the `MAXPIECESIZE` option of the `CONFIGURE CHANNEL` or `ALLOCATE CHANNEL` commands. This option limits backup piece size to the specified umber of bytes. If the total size of the backup set is greater than the specified backup piece size, then multiple physical pieces will be created to hold the backup set contents.

For example, if datafile 1 is 6GB, you can still restrict the backup piece size for disk backups to 2 GB by configuring an automatic disk channel, and then run a backup as follows:

```
CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE = 2G;
```

```
BACKUP AS BACKUPSET DATAFILE 1;
```

A LIST BACKUP command reveals that RMAN created five backup pieces rather than one backup piece to conform to the MAXPIECESIZE size restriction:

```
BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ --------------------
29      Full    9728M      DISK        00:00:35     NOV 02 2002 18:29:26
  List of Datafiles in backup set 29
  File LV Type Ckp SCN    Ckp Time             Name
  ---- -- ---- ---------- -------------------- ----
  1       Full 177590     NOV 02 2002 18:28:51 /oracle/oradata/trgt/system01.dbf

  Backup Set Copy #1 of backup set 29
  Device Type Elapsed Time Completion Time      Tag
  ----------- ------------ -------------------- ---
  DISK        00:00:35     NOV 02 2002 18:29:26 TAG20021102T152701

    List of Backup Pieces for backup set 29 Copy #1
    BP Key  Pc# Status      Piece Name
    ------- --- ----------- ----------
    53      1   AVAILABLE   /oracle/dbs/10d85733_1_1
    54      2   AVAILABLE   /oracle/dbs/10d85733_2_1
    55      3   AVAILABLE   /oracle/dbs/10d85733_3_1
    56      4   AVAILABLE   /oracle/dbs/10d85733_4_1
    57      5   AVAILABLE   /oracle/dbs/10d85733_5_1
```

This option can be used for media managers that cannot manage a backup piece that spans more than one tape. For example, if a tape can hold 10GB, but the backup set being created must hold 80GB of data, then RMAN must be instructed to create backup pieces of 10GB, small enough to fit on the tapes used with the media manager. The backup set media will in this case consist of eight tapes. Media managers supporting SBT2.0 can return a value to RMAN indicating the largest supported backup piece size, which RMAN will use in planning backup activities.

See Also:

- *Oracle Database Recovery Manager Reference* for ALLOCATE CHANNEL syntax
- *Oracle Database Recovery Manager Reference* for CONFIGURE syntax

Number and Size of Backup Sets

Use the *backupSpec* clause of the BACKUP command to specify the objects that you want to back up as well as specify other options. Each *backupSpec* clause produces at least one backup set. The total number and size of backup sets depends for the most part on an internal RMAN algorithm, although you can tune RMAN behavior to a certain extent with the MAXSETSIZE parameter.

Factors Affecting the Number and Size of Backup Sets

In determining the characteristics of the RMAN backup sets, the internal algorithm is influenced by the following factors:

- The number of input files specified in each *backupSpec* clause
- The number of channels that you allocate
- The default number of files in each backup set (4 for datafiles and 16 for archived logs)
- The default number of files read simultaneously by a single channel (8)
- The MAXSETSIZE parameter (specified on the CONFIGURE and BACKUP commands), which specifies a maximum backup set size

The most important rules in the algorithm for backup set creation are:

- Each allocated channel that performs work in the backup job—that is, each channel that is not idle—generates at least one backup set.

> **Note:** RMAN writes backup pieces sequentially; striping a backup piece across multiple output devices is not supported. For example, RMAN does not simultaneously write half of a backup piece to one device and the other half to another device, nor can it write the first piece of a backup set to one device andthe second piece to another device.

- RMAN always tries to divide the backup load so that all allocated channels have roughly the same amount of work to do.
- The default number of datafiles in each backup set is determined by an internal RMAN limit (16 for archived logs, 4 for datafiles).

- The number of datafiles multiplexed in a backup set is limited by the lesser of the number of files in each backup set and the default number of files read by a single channel simultaneously (8).
- The maximum size of a backup set is determined by the MAXSETSIZE parameter of the CONFIGURE or BACKUP command.

> **See Also:** *Oracle Database Recovery Manager Reference* to learn the syntax for the *backupSpec* clause, and Chapter 14, "Tuning Backup and Recovery" to learn about RMAN buffer management

Overview of the MAXSETSIZE Parameter

To specify the maximum size of each backup set, use the MAXSETSIZE parameter in the CONFIGURE or BACKUP command. By limiting the overall size of the backup set, the parameter indirectly limits the number of files in the set and can possibly force RMAN to create additional backup sets.

Specifying MAXSETSIZE: Example Assume that you want to back up 50 datafiles, each containing 1000 blocks. To set the maximum size of each backup set to 10 MB, use the following command:

```
BACKUP DATABASE MAXSETSIZE = 10M;
```

> **Caution:** If a datafile being backed up is bigger than MAXSETSIZE then your backup will fail. Always ensure that MAXSETSIZE is as large as your largest datafile.

> **See Also:** *Oracle Database Recovery Manager Reference* for information on the MAXSETSIZE parameter

I/O Read Rate of Backups

By default, RMAN uses all available I/O bandwidth to read/write to disk. You can limit the I/O resources consumed by a backup job with the RATE option of the ALLOCATE CHANNEL or CONFIGURE CHANNEL commands. The RATE option specifies the maximum number of bytes for each second that RMAN reads on the channel.

For example, you can configure automatic channels to limit each channel to read 1 MB a second:

```
CONFIGURE DEVICE TYPE sbt PARALLELISM 2;
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

```
CONFIGURE CHANNEL DEVICE TYPE sbt RATE 1M;
```

In effect, the RATE option throttles RMAN so that a backup job does not consume excessive I/O bandwidth on the computer.

> **See Also:** "Tuning RMAN Backup Performance: Examples" on page 14-8 for tips about how to optimize RMAN performance

RMAN Backup Types

As explained in Table 2–1, RMAN backups can be classified in these ways:

- Full or incremental
- Open or closed
- Consistent or inconsistent

Note that these backup classifications apply only to datafile backups. Backups of other files, such as archivelogs and control files, always include the complete file and are never inconsistent.

Table 2–1 Backup Types

Backup Type	Definition
Full	A backup that includes every block in the file being backed up, except never-used blocks omitted due to unused block compression. A full backup cannot be part of an incremental backup strategy; it cannot be the parent for a subsequent incremental backup **Note:** A full backup is different from a **whole database backup**, which is a backup of all datafiles and the current control file.
Incremental	An incremental backup is either a level 0 backup, which includes every block in the file except blocks compressed out because they have never been used, or a level 1 backup, which includes only those blocks that have been changed since the parent backup was taken. A level 0 incremental backup is physically identical to a full backup. The only difference is that the level 0 backup can be used as the parent for a level 1 backup, but a full backup will never be used as the parent for a level 1 backup.
Open	A backup of online, read/write datafiles when the database is open.
Closed	A backup of any part of the target database when it is mounted but not open. Closed backups can be consistent or inconsistent.

Table 2–1 Backup Types

Backup Type	Definition
Consistent	A backup taken when the database is mounted (but not open) after a normal shutdown. The checkpoint SCNs in the datafile headers match the header information in the control file. None of the datafiles has changes beyond its checkpoint. Consistent backups can be restored without recovery. **Note:** If you restore a consistent backup and open the database in read/write mode without recovery, transactions after the backup are lost. You still need to perform an `OPEN RESETLOGS`.
Inconsistent	A backup of any part of the target database when it is open or when a crash occurred or `SHUTDOWN ABORT` was run prior to mounting. An inconsistent backup requires recovery to become consistent.

Incremental Backups

The goal of an incremental backup is to back up only those data blocks that have changed since a previous backup. You can use RMAN to create incremental backups of datafiles, tablespaces, or the whole database.

During media recovery, RMAN examines the restored files to determine whether it can recover them with an incremental backup. If it has a choice, then RMAN always chooses incremental backups over archived logs, as applying changes at a block level is faster than reapplying individual changes.

RMAN does not need to restore a base incremental backup of a datafile in order to apply incremental backups to the datafile during recovery. For example, you can restore non-incremental image copies of the datafiles in the database, and RMAN can recover them with incremental backups.

Incremental backups allow faster daily backups, use less network bandwidth when backing up over a network, and provide better performance when tape I/O bandwidth limits backup performance. They also allow recovery of database changes not reflected in the redo logs, such as direct load inserts. Finally, incremental backups can be used to back up `NOARCHIVELOG` databases, and are smaller than complete copies of the database (though they still require a clean database shutdown).

One effective strategy is to make incremental backups to disk (as image copies), and then back up these image copies to a media manager with `BACKUP AS BACKUPSET`. Then, you do not have the problem of keeping the tape streaming that sometimes occurs when making incremental backups directly to tape. Because incremental backups are not as big as full backups, you can create them on disk more easily.

Incremental Backup Algorithm

Each data block in a datafile contains a system change number (SCN), which is the SCN at which the most recent change was made to the block. During an incremental backup, RMAN reads the SCN of each data block in the input file and compares it to the checkpoint SCN of the parent incremental backup. (If block change tracking is enabled, RMAN does not read the portions of the file known to have not changed since the parent incremental backup.) If the SCN in the input data block is greater than or equal to the checkpoint SCN of the parent, then RMAN copies the block.

One consequence of this mechanism is that RMAN applies all blocks containing changed data during recovery—even if the change is to an object created with the `NOLOGGING` option. Hence, making incremental backups is a safeguard against the loss of changes made by `NOLOGGING` operations.

> **See Also:** *Oracle Database Concepts* for more information about `NOLOGGING` mode

Multilevel Incremental Backups

RMAN can create **multilevel incremental backups**. Each incremental level is denoted by a value of 0 or 1. A level 0 incremental backup, which is the base for subsequent incremental backups, copies all blocks containing data. The only difference between a level 0 incremental backup and a full backup is that a full backup is never included in an incremental strategy.

A level 1 incremental backup can be either of the following types:

- A **differential backup**, which backs up all blocks changed after the most recent incremental backup at level 1 or 0
- A **cumulative backup**, which backs up all blocks changed after the most recent incremental backup at level 0

Incremental backups are differential by default.

> **Note:** Cumulative backups are preferable to differential backups when recovery time is more important than disk space, because fewer incremental backups need to be applied during recovery.

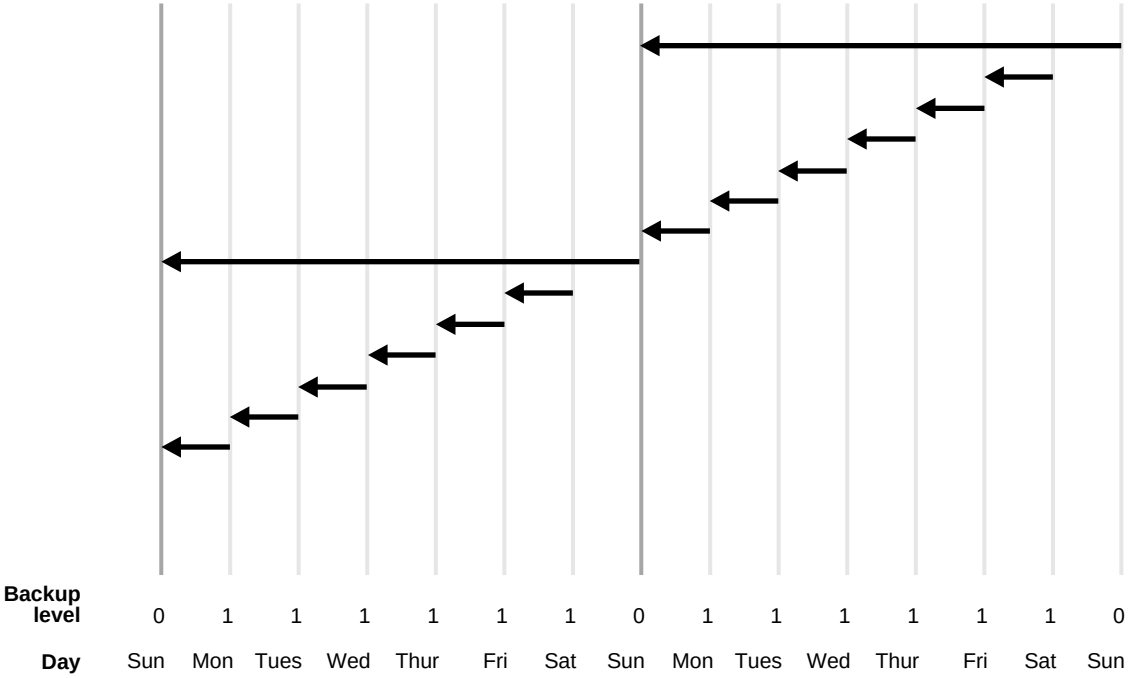
The size of the backup file depends solely upon the number of blocks modified and the incremental backup level.

Differential Incremental Backups

In a differential level 1 backup, RMAN backs up all blocks that have changed since the most recent incremental backup at level 1 (cumulative or differential) or level 0. For example, in a differential level 1 backup, RMAN determines which level 1 backup occurred most recently and backs up all blocks modified after that backup. If no level 1 is available, RMAN copies all blocks changed since the base level 0 backup.

If no level 0 backup is available, then the behavior varies with the compatibility mode setting. If compatibility is >=10.0.0, RMAN copies all blocks that have been changed since the file was created. Otherwise, RMAN behaves as it did in previous releases, by generating a level 0 backup.

Figure 2–4 Differential Incremental Backups



In the example shown in Figure 2–4, the following occurs each week:

- Sunday

An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.

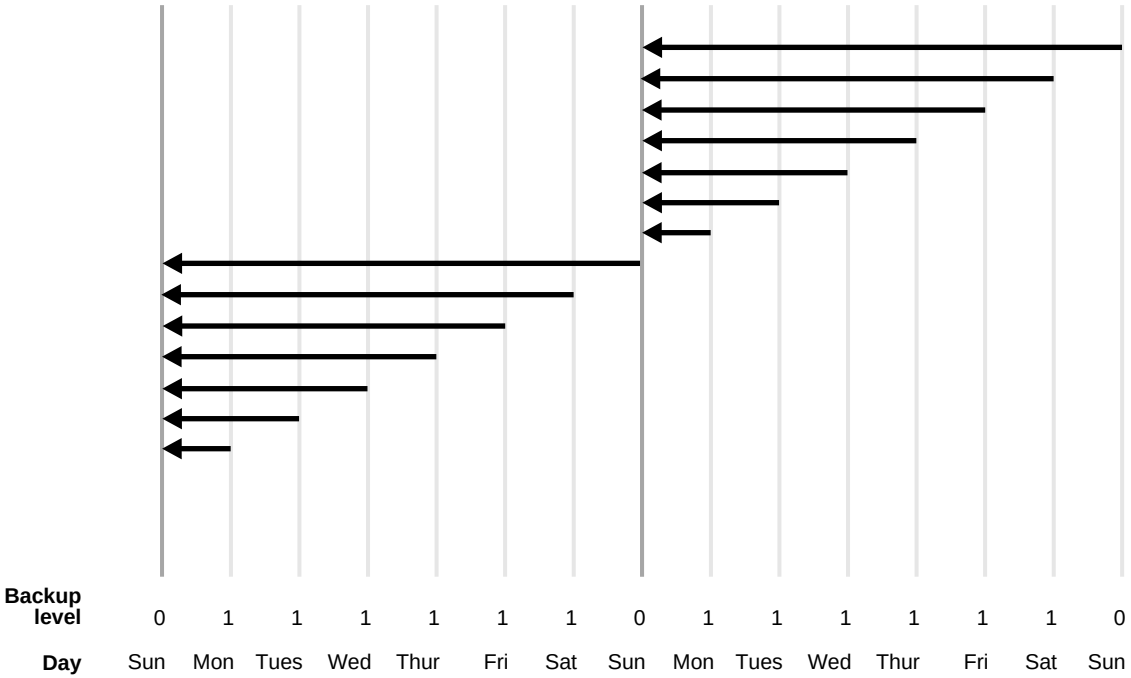
- Monday - Saturday

  On each day from Monday through Saturday, a differential incremental level 1 backup backs up all blocks that have changed since the most recent incremental backup at level 1 or 0. The Monday backup copies blocks changed since Sunday level 0 backup, the Tuesday backup copies blocks changed since the Monday level 1 backup, and so forth.

Cumulative Incremental Backups

In a cumulative level 1 backup, RMAN backs up all the blocks used since the most recent level 0 incremental backup. Cumulative incremental backups reduce the work needed for a restore by ensuring that you only need one incremental backup from any particular level. Cumulative backups require more space and time than differential backups, however, because they duplicate the work done by previous backups at the same level.

Figure 2–5 Cumulative Incremental Backups



In the example shown in Figure 2–5, the following occurs each week:

- Sunday

  An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.

- Monday - Saturday

  A cumulative incremental level 1 backup copies all blocks changed since the most recent level 0 backup. Because the most recent level 0 backup was created on Sunday, the level 1 backup on each day Monday through Saturday backs up all blocks changed since the Sunday backup.

Planning an Incremental Backup Strategy

Choose a backup scheme according to an acceptable MTTR (mean time to recover). For example, you can implement a three-level backup scheme so that a full or level 0 backup is taken monthly, a cumulative level 1 is taken weekly, and a differential

level 1 is taken daily. In this scheme, you never have to apply more than a day's worth of redo for complete recovery.

When deciding how often to take full or level 0 backups, a good rule of thumb is to take a new level 0 whenever 20% or more of the data has changed. If the rate of change to your database is predictable, then you can observe the size of your incremental backups to determine when a new level 0 is appropriate. The following query displays the number of blocks written to a backup set for each datafile with at least 50% of its blocks backed up:

```
SELECT FILE#, INCREMENTAL_LEVEL, COMPLETION_TIME, BLOCKS, DATAFILE_BLOCKS
  FROM V$BACKUP_DATAFILE
  WHERE INCREMENTAL_LEVEL > 0
  AND BLOCKS / DATAFILE_BLOCKS > .5
  ORDER BY COMPLETION_TIME;
```

Compare the number of blocks in differential or cumulative backups to a base level 0 backup. For example, if you only create level 1 cumulative backups, then take a new level 0 backup when the most recent level 1 backup is about half of the size of the base level 0 backup.

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how make incremental backups

Control File and Server Parameter File Autobackups

Having recent backups of your control file and server parameter file is extremely valuable in many recovery situations. To increase the likelihood that you will have such backups, the Oracle database supports control file and server paramter file autobackups. RMAN can automatically back up the control file and server parameter file (SPFILE) in situations in which the RMAN repository data for your database has been updated in a way that affects RMAN's ability to restore your database.

With a control file autobackup, RMAN can recover the database even if the current control file, recovery catalog, and server parameter file are inaccessible. Because the path used to store the autobackup follows a well-known format, RMAN can search for and restore the server parameter file from that autobackup.

After you have started the instance with the restored server parameter file, RMAN can restore the control file from the autobackup. After you mount the control file, use the RMAN repository in the mounted control file to restore the datafiles.

A control file autobackup lets you restore the RMAN repository contained in the control file when the control file is lost and you have no recovery catalog. You do not need a recovery catalog or target database control file to restore the control file autobackup. For example, you can issue:

```
RESTORE CONTROLFILE FROM AUTOBACKUP;
```

After you restore and mount the control file, you have the backup information necessary to restore and recover the database. You can connect to the target instance in NOCATALOG mode and recover the database. You can also create a new recovery catalog and register the target database. The RMAN repository records in the control file will be copied to the new recovery catalog.

The automatic backup of the control file occurs independently of any backup of the current control file explicitly requested as part of your backup command. You can turn the autobackup feature on or off by running the following commands:

```
CONFIGURE CONTROLFILE AUTOBACKUP ON;
CONFIGURE CONTROLFILE AUTOBACKUP OFF;
```

Oracle Corporation recommends that `CONFIGURE CONTROLFILE AUTOBACKUP` be set to `ON`.

How RMAN Performs Control File Autobackups

The first channel allocated during the backup job creates the autobackup and places it into its own backup set; for autobackups after database structural changes, the default disk channel makes the backup. If a server parameter file is used, it is backed up in the same backup set as the control file during a control file autobackup.

After the control file autobackup completes, the database writes a message containing the complete path of the backup piece and the device type to the alert log.

The RMAN behavior when the `BACKUP` command includes datafile `1` depends on the `CONFIGURE CONTROLFILE AUTOBACKUP` setting. If control file autobackups are ON and the backup includes datafile 1, RMAN writes the control file and SPFILE to a separate autobackup backup set. If control file autobackups are OFF and the backup includes datafile 1, then RMAN includes the current control file and SPFILE in the same backup set as the datafiles.

The control file autobackup filename has a default format of `%F` for all device types, so that RMAN can guess the file location and restore it without a repository. The

substitution variable %F is defined in the description of the CONFIGURE command in *Oracle Database Backup and Recovery Basics*. You can specify a different format with the CONFIGURE CONTROLFILE AUTOBACKUP FORMAT command. All autobackup formats must include the %F variable.

The SET CONTROLFILE AUTOBACKUP FORMAT command, which you can specify either within a RUN block or at the RMAN prompt, overrides the configured autobackup format in the session only. The order of precedence is:

1. SET within a RUN block
2. SET at RMAN prompt
3. CONFIGURE CONTROLFILE AUTOBACKUP FORMAT

You can configure the autobackup format even when CONFIGURE CONTROLFILE AUTOBACKUP is set to OFF, but RMAN does not generate autobackups in this case. For RMAN to make autobackups, you must set CONFIGURE CONTROLFILE AUTOBACKUP to ON.

See Also:

- *Oracle Database Recovery Manager Reference* for BACKUP syntax
- *Oracle Database Recovery Manager Reference*for RESTORE syntax

When RMAN Performs Control File Autobackups

By default, control file autobackups are turned off, and no control file autobackups are performed. If CONFIGURE CONTROLFILE AUTOBACKUP is ON, then RMAN automatically backs up the control file and the current server parameter file (if used to start up the database) in one of two circumstances: when a successful backup must be recorded in the RMAN repository, and when a structural change to the database affects the contents of the control file which therefore must be backed up.

Control File Autobackups After Backup Acivities

This means that the control file is backed up in the following situations:

- After every BACKUP command issued at the RMAN prompt.
- At the end of a RUN block, if the last command in the block was BACKUP.
- Whenever a BACKUP command within a RUN block is followed by a command that is not BACKUP.

The first channel allocated during the backup job creates the autobackup and places it into its own backup set. The control file autobackup may be written to disk or tape.

Control File Autobackups After Database Structural Changes

The control file is also automatically backed up after database structural changes such as adding a new tablespace, altering the state of a tablespace or datafile (for example, bringing it online), adding a new online redo log, renaming a file, adding a new redo thread, and so on. Losing this information would compromise your ability to recover the database.

This backup is performed by the server process itself, rather than one of the RMAN channels. This type of autobackup, unlike autobackups that occur after a successful backup, is always created on disk. You can use `CONFIGURE CONTROLFILE AUTOBACKUP FOR DEVICE TYPE DISK` to set the location for this disk based control file autobackup. Note that a failure of the automatic control file autobackup after a structural change never causes the associated structural change to fail. For example, if you add a datafile, and if the resulting control file autobackup fails, then the datafile addition is still successful.

Backup Retention Policies

You can use the `CONFIGURE RETENTION POLICY` command to create a persistent and automatic **backup retention policy**. When a backup retention policy is in effect, RMAN considers backups of datafiles and control files as **obsolete**, that is, no longer needed for recovery, according to criteria that you specify in the `CONFIGURE` command. You can then use the `REPORT OBSOLETE` command to view obsolete files and `DELETE OBSOLETE` to delete them.

As you produce backups over time, older backups become obsolete as they are no longer needed to satisfy the retention policy. RMAN can identify the obsolete files for you, but it does not automatically delete them. You must use the `DELETE OBSOLETE` command to delete files that are no longer needed to satisfy the retention policy.

If you have a flash recovery area configured, however, then the database automatically deletes unnecessary files from the flash recovery area based on its internal disk quota rules. The disk quota rules are distinct from the backup retention policy rules, but the database will never delete files in violation of the retention policy to satisfy the disk quota.

The term **obsolete** does not mean the same as **expired**. A backup is obsolete when `REPORT OBSOLETE` or `DELETE OBSOLETE` determines, based on the user-defined retention policy, that it is not needed for recovery. A backup is considered expired only when RMAN performs a crosscheck and cannot find the file. In short, *obsolete* means "not needed," whereas *expired* means "not found."

From the perspective of a retention policy, a datafile backup is a full or level 0 backup of an individual datafile or control file. It does not matter whether the backup is a datafile image copy, a proxy copy, or part of a backup set. For datafile copies and proxy copies, if RMAN determines that the copy or proxy copy is not needed, then the copy or proxy copy can be deleted. For datafile backups in backup sets, RMAN cannot delete the backup set until all of the individual datafile backups within the backup set are obsolete.

Besides affecting full or level 0 datafile and control file backups, the retention policy affects archived redo logs and level 1 incremental backups. First, RMAN decides which datafile and control file backups are obsolete. Then, RMAN considers as obsolete all archived logs and incremental level 1 backups that are not needed to recover the oldest datafile or control file backup that must be retained.

> **Note:** RMAN cannot implement an automatic retention policy if backups are deleted by non-RMAN methods, for example, through the media manager's tape retention policy. The media manager should never expire a tape until all RMAN backups on that tape have been removed from the media manager's catalog.

There are two mutually exclusive options for implementing a retention policy: **redundancy** and **recovery window**. If no retention policy is configured by the user, then the `REPORT OBSOLETE` and `DELETE OBSOLETE` commands use a default retention policy of `REDUNDANCY 1`.

To configure a retention policy based on a recovery window, use the following command:

- `CONFIGURE RETENTION POLICY TO RECOVERY WINDOW`

To configure a retention policy based on redundancy, use the following command:

- `CONFIGURE RETENTION POLICY TO REDUNDANCY`

You can also disable the retention policy completely, meaning that RMAN does not consider any backup to be obsolete. To do so, use the following command:

```
CONFIGURE RETENTION POLICY TO NONE;
```

Recovery Window

A recovery window is a period of time that begins with the current time and extends backward in time to the **point of recoverability**. The point of recoverability is the earliest time for a hypothetical point-in-time recovery, that is, the earliest point to which you can recover following a media failure. For example, if you implement a recovery window of one week, then this window of time must extend back exactly seven days from the present so that you can restore a backup and recover it to this point. You implement this retention policy as follows:

```
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

This command ensures that for each datafile one backup that is older than the point of recoverability must be retained. For example, if the recovery window is 7, then there must always exist one backup of each datafile that satisfies the following condition:

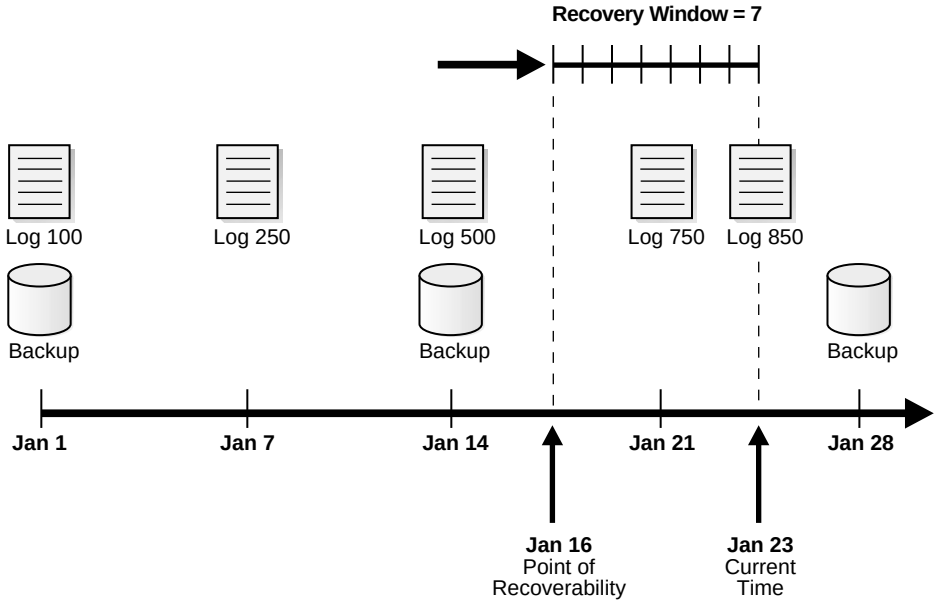
```
SYSDATE - BACKUP CHECKPOINT TIME >=  7
```

All backups older than the most recent backup that satisfied this condition are obsolete.

Assume the following retention policy illustrated in Figure 2–6. The retention policy has the following aspects:

- The recovery window is 7 days.
- Database backups are scheduled every two weeks on these days:
  - January 1
  - January 15
  - January 29
  - February 12
- The database runs in `ARCHIVELOG` mode, and archived logs are saved on disk only as long as needed for the retention policy.

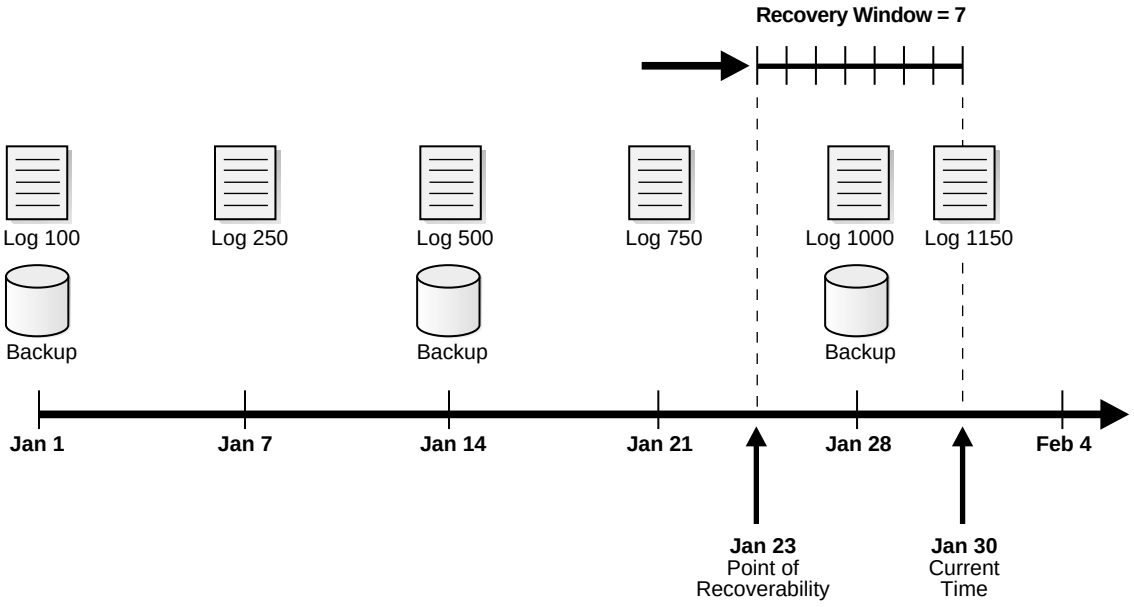
Figure 2–6 Recovery Window, Part 1



As illustrated in Figure 2–6, the current time is January 23 and the point of recoverability is January 16. Hence, the January 14 backup is needed for recovery, and so are the archived logs from log sequence 500 through 850. The logs before 500 and the January 1 backup are obsolete because they are not needed for recovery to a point within the window.

Assume the same scenario a week later, as depicted in Figure 2–7.

Figure 2–7 Recovery Window, Part 2



In this scenario, the current time is January 30 and the point of recoverability is January 23. Note how the January 14 backup is *not* obsolete even though a more recent backup (January 28) exists in the recovery window. This situation occurs because restoring the January 28 backup does not enable you to recover to the earliest time in the window, January 23. To ensure recoverability to any point within the window, you must save the January 14 backup as well as all archived redo logs from log sequence 500 to 1150.

Backup Redundancy

A redundancy-based retention policy specifies how many backups of each datafile must be retained. For example, you can specify:

```
CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
```

Although the recovery window is the best practice for specifying a retention policy, it can complicate disk space usage planning because the number of backups that must be kept by the recovery window is not constant and depends on the backup schedule. Use the `CONFIGURE RETENTION POLICY TO REDUNDANCY` *n* command to implement a retention policy that maintains a constant number of backups of

each datafile. RECOVERY WINDOW and REDUNDANCY-based retention policies are mutually exclusive.

The default retention policy is REDUNDANCY = 1, to maintain compatibility with the behavior of REPORT OBSOLETE in earlier RMAN releases. You can also run the following command to disable the retention policy altogether:

```
CONFIGURE RETENTION POLICY TO NONE;
```

If the retention policy is configured to NONE, then REPORT OBSOLETE and DELETE OBSOLETE do not consider any backups to be obsolete.

Batch Deletes of Obsolete Backups

Run the REPORT OBSOLETE command to determine which backups are currently obsolete according to the retention policy.

If you configure the retention policy to NONE, then RMAN does not consider any backups as obsolete. Consequently, RMAN issues an error when you run REPORT OBSOLETE without any other options and the retention policy is set to NONE.

A companion command, DELETE OBSOLETE, deletes all files which are obsolete according to the retention policy. You should run DELETE OBSOLETE periodically to minimize space wasted by storing obsolete backups. For example, you can run DELETE OBSOLETE in a weekly script.

You can also specify the REDUNDANCY or RECOVERY WINDOW options on the REPORT or DELETE commands, to overrride the configured retention policy.

The REPORT OBSOLETE and DELETE OBSOLETE commands work in two steps:

1. For each datafile for which there are full backup, datafile copy, or level 0 incremental backups, RMAN identifies the oldest full or level 0 backup or copy that is not obsolete under the retention policy being tested. Any full backup, level 0 incremental backup, or datafile copy of a datafile older than the one identified in this step is considered obsolete.
2. Any archived logs and level 1 incremental backups that are older than the oldest non-obsolete full backup are then obsolete because there is no full or level 0 backup to which they can be applied.

See Also:

- *Oracle Database Backup and Recovery Basics* to generate reports and delete backups
- *Oracle Database Recovery Manager Reference* for DELETE syntax
- *Oracle Database Recovery Manager Reference* for REPORT syntax

Exempting Backups from the Retention Policy

You may want to store a **long-term backup**, potentially offsite, for much longer than the time dictated by the retention policy. For example, you may make a database backup on the first day of every year to satisfy some regulatory requirement, independent of backups taken for your ongoing backup and recovery strategy.

Such long-term backups should be recorded in the RMAN repository, but they must be exempted from the retention policy because RMAN would quickly consider them obsolete, and they would be removed the next time the DELETE OBSOLETE command is used.

You can exempt a backup from the retention policy by using the KEEP option with the BACKUP command when you create the backup, or the KEEP option of the CHANGE command to exempt an existing backup. Note that backups exempted from the retention policy are still fully valid backups, which can be used in restore and recovery operations like any other if RMAN judges them to be the best choice available.

You can change the exempt status of a backup using the CHANGE... KEEP and CHANGE... NOKEEP commands. The NOKEEP option (default) indicates that the backup is not immune from the configured retention policy.

You can specify the LOGS option to save archived logs for a possible incomplete recovery of the long-term backup. When LOGS is specified, all logs more recent than the backup are kept as long as the backup is kept. In other words, KEEP UNTIL TIME... LOGS means that RMAN will keep all logs required to recover the backup as long as the backup is kept. If you specify NOLOGS, then RMAN does not keep the logs required to recover the backup. Note that if you use KEEP UNTIL TIME... with an inconsistent backup, you must use the LOGS option, or that backup will become unusable when the logs required to recover it are deleted as obsolete.

You can specify an end date using the UNTIL clause, or either specify that the backup should be kept FOREVER. If you specify UNTIL, then RMAN will not mark the backup as obsolete until after the UNTIL date has passed. Note that it is an error

to specify KEEP FOREVER with the LOGS option, as this would require keeping all redo logs forever.

The following commands are examples of long-term backups:

```
# Creates a backup and exempts it from retention policy until last day of 2003
BACKUP DATABASE KEEP UNTIL TIME "TO_DATE('31-DEC-2003', 'dd-mon-yyyy')" NOLOGS;

# Specifies that backupset 231 is no longer exempt from the retention policy
CHANGE BACKUPSET 231 NOKEEP;

# Creates a backup that is indefinitely exempt from the retention policy
BACKUP TABLESPACE users KEEP FOREVER NOLOGS;
```

> **See Also:** *Oracle Database Recovery Manager Reference* for CHANGE syntax

Relationship Between Retention Policy and Flash Recovery Area Rules

The RMAN status OBSOLETE is always determined in reference to a retention policy. For example, if a database backup is OBSOLETE in the RMAN repository, it is because it is either not needed for recovery to a point within the recovery window, or it is redundant.

If you configure a flash recovery area, then the database uses an internal algorithm to delete files from the flash recovery area that are no longer needed because they are redundant, orphaned, and so forth. The backups with status OBSOLETE form a subset of the files deemed eligible for deletion by the disk quota rules.

There is one important difference between the flash recovery area criteria for OBSOLETE status and the disk quota rules for deletion eligibility. Assume that archived logs 1000 through 2000, which are on disk, are needed for the currently enabled recovery window and so are not obsolete. If you back up these logs to tape, then the retention policy still considers the disk logs as required, that is, not obsolete. Nevertheless, the flash recovery area disk quota algorithm considers the logs on disk as eligible for deletion because they have already been backed up to tape. The logs on disk do not have OBSOLETE status in the repository, yet are eligible for deletion by the flash recovery area. Note, though, that the retention policy is never violated when determining which files to delete from the flash recovery area.

Backup Optimization

Backup optimization is a feature that avoids creating identical backup copies of files that have not changed since the last time they were backed up. If you enable backup optimization, then the BACKUP command skips the backup of a file when the identical file has already been backed up to the allocated device type.

Backup Optimization Algorithm

Table 2–3 describes criteria that RMAN uses to determine whether a file is identical to a file that it already backed up.

Table 2–2 Criteria to Determine an Identical File

Type of File	Criteria to Determine an Identical File
Datafile	Same DBID, checkpoint SCN, creation SCN, and RESETLOGS SCN and time. The datafile must be offline-normal, read-only, or closed normally.
Archived redo log	Same thread, sequence number, and RESETLOGS SCN and time.
Backup set	Same backup set recid and stamp.

If RMAN determines that a file is identical and it has already been backed up, then it is a candidate to be skipped. However, RMAN must do further checking to determine whether to skip the file, because both the retention policy and the backup duplexing feature are factors in the algorithm that determines whether RMAN has sufficient backups on the specified device type.

Table 2–3 describes the algorithm that backup optimization uses when determining whether to skip the backup of an identical file.

Table 2–3 Backup Optimization Algorithm

For an Identical ...	Backup Optimization Algorithm
Datafile	If you configured a recovery window, then RMAN skips a datafile backup *only if* the latest backup of a file is earlier than the earliest point in the window. In other words, RMAN takes another backup of a file, ignoring any possible optimization, if the latest file was backed up longer ago than the recovery window. This is done to allow media to be recycled after the media expires. This is not done for device type DISK. If you did not configure a recovery window, then RMAN sets *r*=1 by default and searches for values of *n* in this order of precedence (that is, values higher on the list override values lower on the list): **1.** If `CONFIGURE RETENTION POLICY TO REDUNDANCY` *r* is enabled, then RMAN only skips datafiles when *n*=*r*+1 backups exist. **2.** `BACKUP . . . COPIES` *n* **3.** `SET BACKUP COPIES` *n* **4.** `CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE . . . TO` *n* RMAN skips backup only if at least *n* backups of an identical file exist on the specified device. If RMAN does not skip the backup, then it makes the backup exactly as specified.
Archived log	By default, *n*=1. RMAN searches for values of *n* in this order of precedence (that is, values higher on the list override values lower on the list): **1.** `BACKUP . . . COPIES` *n* **2.** `SET BACKUP COPIES` *n* **3.** `CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE . . . TO` *n* RMAN skips backup only if at least *n* backups of an identical file exist on the specified device. If RMAN does not skip the backup, then it makes the backup exactly as specified.
Backup set	By default, *n*=1. RMAN searches for other values of *n* in this order of precedence (that is, values higher on the list override values lower on the list): **1.** `BACKUP . . . COPIES` *n* **2.** `SET BACKUP COPIES` *n* RMAN skips backup only if at least *n* backups of an identical file exist on the specified device. If RMAN does not skip the backup, then it makes the backup exactly as specified.

For example, assume that at 9 a.m. you back up three copies of all existing archived logs to tape:

```
BACKUP DEVICE TYPE sbt COPIES 3 ARCHIVELOG ALL;
```

Later, you enable the following configuration setting in preparation for another backup:

```
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE sbt TO 4;
CONFIGURE BACKUP OPTIMIZATION ON;
```

Then, you run the following archived log backup at noon:

```
BACKUP DEVICE TYPE sbt COPIES 2 ARCHIVELOG ALL;
```

In this case, the BACKUP . . . COPIES setting overrides the CONFIGURE . . . COPIES setting, so RMAN sets *n*=2. RMAN skips the backup of a log only if at least two copies of the log exist on the sbt device. Because three copies of each log exist on sbt of all the logs generated before 9 a.m., RMAN skips the backups of these logs. However, RMAN backs up two copies of all logs generated after 9 a.m. because these logs have not yet been backed up to tape.

At this point, three copies of the logs created before 9 a.m. exist on tape, and two copies of the logs created after 9 a.m. exist on tape. Assume that you run the following backup at 3 p.m.:

```
RUN
{
  SET BACKUP COPIES 3;
  BACKUP DEVICE TYPE sbt ARCHIVELOG ALL;
}
```

In this case, RMAN sets *n*=3 and so will not back up the logs created before 9 a.m. because three copies already exist on tape. However, only two copies of the logs created after 9 a.m. exist on tape, so RMAN does not optimize backups of these logs. Hence, RMAN backs up three copies of the logs created after 9 a.m.

Requirements for Enabling and Disabling Backup Optimization

Backup optimization is used when the following conditions are true:

- The CONFIGURE BACKUP OPTIMIZATION ON command has been run.
- You run BACKUP DATABASE, BACKUP ARCHIVELOG with ALL or LIKE options, or BACKUP BACKUPSET ALL.
- Only one type of channel is allocated, that is, you do not mix channels of type DISK and sbt.

For example, assume that you run these commands:

```
BACKUP DEVICE TYPE sbt DATABASE PLUS ARCHIVELOG;
```

```
BACKUP DEVICE TYPE sbt BACKUPSET ALL;
```

If none of these files has changed since the last backup, then RMAN does not back up the files again, nor signal an error if it skips all files specified in the command.

To override backup optimization and back up all files whether or not they have changed, specify the FORCE option on the BACKUP command. To disable backup optimization on a persistent basis, use the following command:

```
RMAN> CONFIGURE BACKUP OPTIMIZATION OFF;
```

Effect of Retention Policies on Backup Optimization

The retention policy influences backup optimization. Because the retention policy defaults to REDUNDANCY=1, a retention policy is always in place unless it is explicitly disabled with CONFIGURE RETENTION POLICY TO NONE.

Note: Use caution when enabling backup optimization if you use a media manager with its own internal expiration policy. Run CROSSCHECK periodically to synchronize the RMAN repository with the media manager. Otherwise, RMAN may skip backups due to optimization without recognizing that the media manager has discarded backups stored on tape.

Backup Optimization and a Recovery Window

If optimization is enabled, and if a recovery window retention policy is in effect, then RMAN always backs up datafiles whose most recent backup is older than the recovery window. For example, assume that:

- Today is February 21.
- The recovery window is 7 days.
- The most recent backup of tablespace `tools` to tape is January 3.
- Tablespace `tools` is read-only.

On February 21, when you issue a command to back up tablespace `tools` to tape, RMAN backs it up even though it did not change after the January 3 backup (because it is read-only). RMAN makes the backup because no backup of the tablespace exists within the 7-day recovery window.

This behavior allows the media manager to expire old tapes. Otherwise, the media manager would be forced to keep the January 3 backup of tablespace `tools`

indefinitely. By making a more recent backup of tablespace `tools` on February 21, RMAN allows the media manager to expire the tape containing the obsolete January 3 backup.

Backup Optimization and Redundancy

Assume that you configure a retention policy for redundancy. In this case, RMAN only skips backups of offline or read-only datafiles when there are *r* + 1 backups of the files, where *r* is set in `CONFIGURE RETENTION POLICY TO REDUNDANCY` *r*.

Assume that you enable backup optimization and set the following retention policy:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
```

So, RMAN only skips backups when three identical files are already backed up. Also assume that you have never backed up the `users` tablespace, which is read / write, and that you perform the actions described in Table 2–4 over the course of the week.

Table 2–4 Effect of Redundancy Setting on Backup Optimization

Day	Action	Result	Redundant Backup
Monday	Take tablespace `users` offline normal.		
Tuesday	`BACKUP DATABASE`	The `users` tablespace is backed up.	
Wednesday	`BACKUP DATABASE`	The `users` tablespace is backed up.	
Thursday	`BACKUP DATABASE`	The `users` tablespace is backed up.	Tuesday backup
Friday	`BACKUP DATABASE`	The `users` tablespace is *not* backed up.	Tuesday backup
Saturday	`BACKUP DATABASE`	The `users` tablespace is *not* backed up.	Tuesday backup
Sunday	`DELETE OBSOLETE`	The Tuesday backup is deleted.	
Monday	`BACKUP DATABASE`	The `users` tablespace is backed up.	Wednesday backup

The backups on Tuesday, Wednesday, and Thursday back up the offline `users` tablespace to satisfy the condition that three backups must exist (one more than redundancy setting). The Friday and Saturday backups do not back up the `users` tablespace because of backup optimization. Note that the Tuesday backup of `users` is obsolete beginning on Thursday.

On Sunday, you delete all obsolete backups, which removes the Tuesday backup of `users`. The Tuesday backup is obsolete because of the retention policy setting. The whole database backup on Monday then backs up the `users` tablespace to satisfy the condition that three backups must exist (one more than redundancy setting). In this way, you can recycle your tapes over time.

> **See Also:** "Backing Up Files Using Backup Optimization" on page 7-9, and "Configuring Backup Optimization" on page 6-21

Restartable Backups

Using the **restartable backups** feature, RMAN can back up only those files that have not been backed up since a specified date. Use this feature after a backup fails to back up the parts of the database missed by the failed backup.

The unit of restartability is a backup set. If the backup generates multiple backup sets, then the backups that completed successfully do not have to be rerun. If the entire database is written into one backup set, and if the backup fails halfway through, then the entire backup has to be restarted.

To take better advantage of restartable backups, you can use set the `MAXSETSIZE` parameter of the `BACKUP` command. If, for instance, you set `MAXSETSIZE` to 10MB for a given backup command, a new backup set is produced for each 10MB of backup output. If the backup fails after some backup sets have been produced and must be restarted, the data backed up in those backup sets will not have to be backed up again. (Note that `MAXSETSIZE` must be large enough that any file can be accomodated in a single backup piece, because large files cannot span backup pieces.)

For example, if the largest datafile is less than 10 MB, then you can back up the database daily as follows:

```
BACKUP DATABASE MAXSETSIZE = 10M;
```

Then, after a failure you can back up all files in the database that were not backed up in the last 24 hours by issuing:

```
BACKUP DATABASE NOT BACKED UP SINCE TIME 'SYSDATE-1';
```

If the SINCE TIME is later than the completion time, then RMAN backs up the file. If you use "BACKUP DATABASE NOT BACKED UP" without the SINCE TIME parameter, then RMAN only backs up files that have never been backed up.

When determining whether a file has been backed up, RMAN compares the SINCE TIME date with the completion time of the most recent backup of the file. The completion time for a backup piece is the completion time of the entire backup set, not an individual backup piece; in other words, all files in the same backup set have the same completion time.

> **See Also:** "Restarting a Backup After It Partially Completes" on page 7-9 and *Oracle Database Recovery Manager Reference* for BACKUP syntax

Managing Backup Windows and Performance: BACKUP... DURATION

A **backup window** is a period of time during which a backup activity must complete. For example, you may want to restrict your database backup activities to a window of time when user activity on your system is low, such as between 2:00 AM and 6:00 AM.

The BACKUP command supports a DURATION argument which lets you specify how long a given backup job is allowed to run. You could, for example, run the following command at 2:00AM:

```
BACKUP DURATION 4:00 TABLESPACE users;
```

RMAN backs up the specified data at the maximum possible speed. If the backup is not complete in four hours, the backup is interrupted. Any completed backupsets are retained and can be used in restore operations, even if the entire backup is not complete. Any incomplete backupsets are discarded.

Controlling RMAN Behavior when Backup Window Ends with PARTIAL

By default, when a BACKUP... DURATION command runs out of time before the backup completes, RMAN reports an error. (The effect of this is that if the command is running in a RUN block, the RUN block terminates.) You can control this behavior by adding the PARTIAL option to the BACKUP... DURATION command, as in this example:

```
BACKUP DURATION 4:00 PARTIAL TABLESPACE users FILESPERSET 1;
```

When PARTIAL is used, no error is reported when a backup command is interrupted due to the end of the backup window. Instead, a message showing which files could not be backed will be displayed. If the BACKUP command is part of a RUN block, then the remaining commands in the RUN block will continue to execute.

When using DURATION the least recently backed up files are backed up first. Thus, if you retry a job that was interrupted when the available duration expired, each successive attempt covers more of the files needing backup.

Note also the use of FILESPERSET 1 in this example. With this option, RMAN puts each file into its own backupset. This way, when a backup is interrupted at the end of the backup window, only the backup of the file currently being backed up is lost. All backup sets completed during the window are saved, minimizing the lost work due to the end of the backup window.

Managing Backup Performance with MINIMIZE TIME and MINIMIZE LOAD

When using DURATION you can run the backup with the maximum possible performance, or run as slowly as possible while still finishing within the allotted time, to minimize the performance impact of backup tasks. To maximize performance, use the MINIMIZE TIME option with DURATION, as shown in this example:

```
BACKUP DURATION 4:00 PARTIAL MINIMIZE TIME DATABASE FILESPERSET 1;
```

To extend the backup to use the full time available, use the MINIMIZE LOAD option, as in this example:

```
BACKUP DURATION 4:00 PARTIAL MINIMIZE LOAD DATABASE FILESPERSET 1;
```

RMAN monitors the progress of the running backup, and periodically estimates how long the backup will take to complete at its present rate. If RMAN estimates that the backup will finish before the end of the backup window, it slows down the

rate of backup so that the full available duration will be used. This reduces the overhead on the database associated with the backup.

Note: Note these issues when using DURATION and MINIMIZE LOAD with a tape backup:

- Efficient backup to tape requires tape streaming. If you use MINIMIZE LOAD, RMAN may reduce the rate of backup to the point where tape streaming is not optimal.
- RMAN will hold the tape resource for the entire duration of the backup window. This prevents the use of the tape resource for any other purpose during the backup window.

Because of these concerns, it is not recommended that MINIMIZE LOAD be used with tape. See "Factors Affecting Backup Speed to Tape" on page 14-6 for more details on efficient tape handling.

RMAN Backup Errors

RMAN detects and responds to two primary types of backup errors: I/O errors and corrupt blocks. Any I/O errors that RMAN encounters when reading files or writing to the backup pieces or image copies cause RMAN to terminate the backup jobs. For example, if RMAN tries to back up a datafile but the datafile is not on disk, then RMAN terminates the backup. If multiple channels are being used, or redundant copies of backups are being created, RMAN may be able to continue the backup without user intervention.

If BACKUP AS BACKUPSET creates more than one complete backup set and an error occurs, then RMAN needs to rewrite the backup sets that it was writing at the time of the error. However, it retains any backup sets that it successfully wrote before terminating. The NOT BACKED UP SINCE option of the BACKUP command restarts a backup that partially completed, backing up only files that did not get backed up.

RMAN copies datafile blocks that are already identified as corrupt into the backup. If RMAN encounters datafile blocks that have not already been identified as corrupt, then RMAN stops the backup unless SET MAXCORRUPT has been used. Setting MAXCORRUPT allows a specified number of previously undetected block corruptions in datafiles during the execution of an RMAN BACKUP command. If RMAN detects more than this number of corruptions while taking the backup, then the command terminates. The default limit is zero, meaning that RMAN does not tolerate corrupt blocks by default.

When RMAN finds corrupt blocks, until it finds enough to exceed the MAXCORRUPT limit, it writes the corrupt blocks to the backup with a reformatted header indicating that the block has media corruption. If the backup completes without exceeding MAXCORRUPT, then the database records the address of the corrupt blocks and the type of corruption in the control file. Access these records through the V$DATABASE_BLOCK_CORRUPTION view. Note that if more than MAXCORRUPT corrupt blocks are found, the V$DATABASE_BLOCK_CORRUPTION view is not populated. In such a case, you should set MAXCORRUPT higher and re-run the command to identify the corrupt blocks.

See Also:

- "Tests and Integrity Checks for Backups" on page 2-58 for more information about fractured and corrupt blocks
- "Restartable Backups" on page 2-54 for more information about the NOT BACKED UP SINCE clause
- *Oracle Database Reference* for a description of V$DATABASE_BLOCK_CORRUPTION
- *Oracle Database Recovery Manager Reference* for SET MAXCORRUPT syntax

Tests and Integrity Checks for Backups

The database prevents operations that result in unusable backup files or corrupt restored datafiles. The database server automatically does the following:

- Blocks access to datafiles while they are being restored or recovered
- Allows only one restore operation for each datafile at a time
- Ensures that incremental backups are applied in the correct order
- Stores information in backup files to allow detection of corruption

You can use the BACKUP VALIDATE and RESTORE VALIDATE commands to test backup and restore operations without creating output files. In this way, you can check your datafiles for possible problems. If you run RMAN with the following configuration, then it detects all types of corruption that are possible to detect:

- In the initialization parameter file, set DB_BLOCK_CHECKSUM=TRUE
- In the RMAN BACKUP and RESTORE commands, do *not* specify the MAXCORRUPT option, do *not* specify the NOCHECKSUM option, but *do* specify the CHECK LOGICAL option

See *Oracle Database Backup and Recovery Basics* for more details on BACKUP VALIDATE and RESTORE VALIDATE.

Detecting Physical and Logical Block Corruption

Because an database server session is performing the backups and has a great understanding of files being backed up or copied, the server session is able to detect many types of physically corrupt blocks during the backup process. Each new corrupt block not previously encountered in a backup is recorded in the control file and in the alert.log. By default, error checking for physical corruption is enabled.

At the end of a backup, RMAN stores the corruption information in the recovery catalog and control file. Access this data using the V$DATABASE_BLOCK_CORRUPTION view.

If the server session encounters a datafile block during a backup that has already been identified as corrupt by the database, then the server session copies the corrupt block into the backup and the corrupt block is recorded the control file as either a logical or media corruption. RMAN copies the block in case the user wants to try to salvage the contents of the block.

If RMAN encounters a datafile block that has media corruption but that has not already been identified as corrupt by the database, then it writes the block to the backup with a reformatted header indicating that the block has media corruption.

Detection of Logical Block Corruption

Besides testing for media corruption, the database can also test data and index blocks for logical corruption, such as corruption of a row piece or index entry. If RMAN finds logical corruption, then it logs the block in the alert.log. If CHECK LOGICAL was used, the block is also logged in the server session trace file. By default, error checking for logical corruption is disabled.

For BACKUP commands the MAXCORRUPT parameter sets the total number of physical and logical corruptions permitted in a file. If the sum of physical and logical corruptions for a file is less than its MAXCORRUPT setting, the RMAN command completes successfully. If MAXCORRUPT is exceeded, the command terminates and RMAN does not read the rest of the file. V$DATABASE_BLOCK_CORRUPTION is populated with corrupt block ranges if the command succeeds. Otherwise, you must set MAXCORRUPT higher and re-run the backup to find out the corrupt block ranges.

See Also: *Oracle Database Recovery Manager Reference* for BACKUP . . . MAXCORRUPT syntax

Detection of Fractured Blocks During Open Backups

One danger in making online backups is the possibility of inconsistent data within a block. For example, assume that you are backing up block 100 in datafile `users.dbf`. Also, assume that the copy utility reads the entire block while database writer is in the middle of updating the block. In this case, the copy utility may read the old data in the top half of the block and the new data in the bottom top half of the block. In this case, the block is a **fractured block**, meaning that the data contained in this block is not consistent.

When performing open backups *without* using RMAN, you must put tablespaces in **backup mode** in case the copy utility reads a block for a backup that is currently being written by the database writer. When not in backup mode, the database records only changed bytes in the redo stream. When a tablespace is in backup mode, each time a block is changed the datbase writes the before-image ofthe entire block to the redo log before modifying it. Then, the database also records the changes to the block in the redo log. When you recover using SQL*Plus, the database applies both the block images and the changes from the redo logs during recovery. Applying the block images repairs any possible fractured blocks in the backup being restored and recovered.

RMAN does not require that you use backup mode (and it is an error to use backup mode with RMAN). During an RMAN backup, a database server session reads each block of the datafile and checks whether each block is fractured by comparing the block header and footer. If a block is fractured, the session re-reads the block. If the same fracture is found, then the block is considered permanently corrupt. If MAXCORRUPT is exceeded, the backup stops.

Backup Validation with RMAN

You can run the BACKUP . . . VALIDATE command to check datafiles for physical and logical corruption, or to confirm that all database files exist in the correct locations. No backup is taken, but all specified files are scanned to verify that they can be backed up. Here is an example:

```
BACKUP VALIDATE DATABASE ARCHIVELOG ALL;
```

You cannot use the MAXCORRUPT or PROXY parameters with the VALIDATE option.

See Also: *Oracle Database Recovery Manager Reference* for BACKUP syntax and "Validating Backups with RMAN" on page 10 for more details on using BACKUP VALIDATE.

3

RMAN Recovery Concepts

This chapter describes the basic concepts involved in using RMAN to restore, recover, and duplicate databases.

This chapter contains these topics:

Restoring Files with RMAN

Use the RMAN RESTORE command to restore the following types of files from disk or other media:

- Database (all datafiles)
- Tablespaces
- Control files
- Archived redo logs
- Server parameter files

Because a backup set is in a proprietary format, you cannot simply copy it as you would a backup database file created with an operating system utility; you must use the RMAN RESTORE command to extract its contents. In contrast, the database can use image copies created by the RMAN BACKUP AS COPY command without additional processing.

Note: You do not normally restore archived logs because RMAN performs this operation automatically as needed during recovery. You can improve recovery performance, however, by manually restoring backups of archived redo logs that you need for recovery.

See Also: *Oracle Database Recovery Manager Reference* for RESTORE syntax and prerequisites

Mechanics of Restore Operations

RMAN automates the procedure for restoring files. You do not need to go into the operating system, locate the backup that you want to use, and manually copy files into the appropriate directories. When you issue a RESTORE command, RMAN directs a server session to restore the correct backups to either:

- The default location, overwriting the files with the same name currently there
- A new location, which you can specify with the SET NEWNAME command

To restore a datafile, either mount the database or keep it open and take the datafile to be restored offline. When RMAN performs a restore, it creates the restored files as datafile image copies and records them in the repository. The following table describes the behavior of the RESTORE, SET NEWNAME, and SWITCH commands.

Run SET NEWNAME?	RESTORE Behavior	Run SWITCH?
No	RMAN restores the files to their current path names and immediately removes the repository records for the datafile copies created during the restore.	N / A
Yes	RMAN restores the files to the path names specified by SET NEWNAME and does not remove the repository records for the datafile copies created during the restore.	If yes, then RMAN updates the datafile names in the control file to the names of the restored files; if no, then RMAN does not update the filenames in the control file and the restored files become datafile copies.

For example, if you restore datafile `?/oradata/trgt/tools01.dbf` to its default location, then RMAN restores the file `?/oradata/trgt/tools01.dbf` and overwrites any file that it finds with the same filename. If you run a SET NEWNAME command before you restore a file, then RMAN creates a datafile copy with the name that you specify. For example, assume that you run the following commands:

```
SET NEWNAME FOR DATAFILE '?/oradata/trgt/tools01.dbf' TO '/tmp/tools01.dbf';
RESTORE DATAFILE '?/oradata/trgt/tools01.dbf';
```

In this case, RMAN creates a datafile copy of `?/oradata/trgt/tools01.dbf` named `/tmp/tools01.dbf` and records it in the repository. To change the name for datafile `?/oradata/trgt/tools01.dbf` to `/tmp/tools01.dbf` in the control file, run a SWITCH command so that RMAN considers the restored file as the current database file. For example:

```
SWITCH DATAFILE '/tmp/tools01.dbf' TO DATAFILECOPY '?/oradata/trgt/tools01.dbf';
```

The SWITCH command is the RMAN equivalent of the SQL statement ALTER DATABASE RENAME FILE.

> **See Also:** *Oracle Database Recovery Manager Reference* for SET NEWNAME syntax, and *Oracle Database Recovery Manager Reference* for SWITCH syntax

File Selection in Restore Operations

RMAN uses the repository to select the best available backups for use in the restore operation. The most recent backup available, or the most recent backup satisfying any UNTIL clause specified in the RESTORE command, is always the preferred

choice. If two backups are from the same point in time, RMAN prefers to use image copies over backup sets because RMAN can restore more quickly from image copies than from backup sets (especially those stored on tape).

All specifications of the RESTORE command must be satisfied before RMAN restores a backup. Unless limited by the DEVICE TYPE clause, the RESTORE command searches for backups on all device types of configured channels.

If no available backup in the repository satisfies all the specified criteria, then RMAN returns an error during the compilation phase of the restore job. If you manually allocate channels, and if the file cannot be restored because no backups exist on the device types allocated in the job, then create a new job specifying channels for devices containing the existing backups. This problem does not occur when you configure automatic channels.

See Also: "Configuring Automatic Channels" on page 6-12

Restore Failover

During a RESTORE operation, if a backup piece, image copy or proxy copy is inaccessible (for instance, deleted from the device) or a block in the backup is corrupted, then RMAN automatically looks for a another usable copy of this backup piece or image copy, on the same device or another device, based on the information in the RMAN repository. If no usable copies are available, then RMAN searches for prior backups. RMAN searches all prior backups for the most recent available backup usable in the current operation until it has exhaused all possibilities.

Restore failover is also used when there are errors restoring archivelogs during RECOVER, BLOCKRECOVER, and FLASHBACK DATABASE commands.

When RMAN performs restore failover to another backup of the same file, you will see a message similar to this one in the output of RMAN:

```
failover to piece handle=/u01/backup/db_1 tag=BACKUP_031009
```

Also, details about block corruptions will be printed in the alert log and trace files.

When restore failover cannot locate another copy of the same backup and searches for a prior backup, the message generated is similar to this example:

```
ORA-19624: operation failed, retry possible
ORA-19505: failed to identify file "/u01/backup/db_1"
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

```
failover to previous backup
```

Restore Optimization

By default, RMAN does not perform a restore if the file to be restored is in the correct place and its header contains the expected information. RMAN only restores a file if the header check does not succeed, although you can use the FORCE option of the RESTORE command to override this behavior and restore the requested files unconditionally.

> **Note:** Restore optimization only checks the datafile header and does not the scan the datafile body for corrupted blocks.

Restore optimization is particularly useful in cases where a restore only partially completes. For example, assume that a full database restore encounters a power failure after all except one of the datafiles has been restored. If you start the same restore again, then RMAN only restores the single datafile that was not restored during the previous attempt.

> **See Also:** *Oracle Real Application Clusters Administrator's Guide* for description of RESTORE behavior in a RAC configuration

Datafile Media Recovery with RMAN

The concept of **datafile media recovery** is the application of online or archived redo logs or incremental backups to a restored datafile in order to update it to the current time or some other specified time. Use the RMAN RECOVER command to perform media recovery and apply logs or incremental backups automatically.

RMAN Media Recovery: Basic Steps

If possible, make the recovery catalog available to perform the media recovery. If it is not available, or if you do not maintain a recovery catalog, then RMAN uses metadata from the target database control file. If both the control file and recovery catalog are lost, then you can still recover the database—assuming that you have backups of the datafiles and at least one autobackup of the control file.

The generic steps for media recovery using RMAN are as follows:

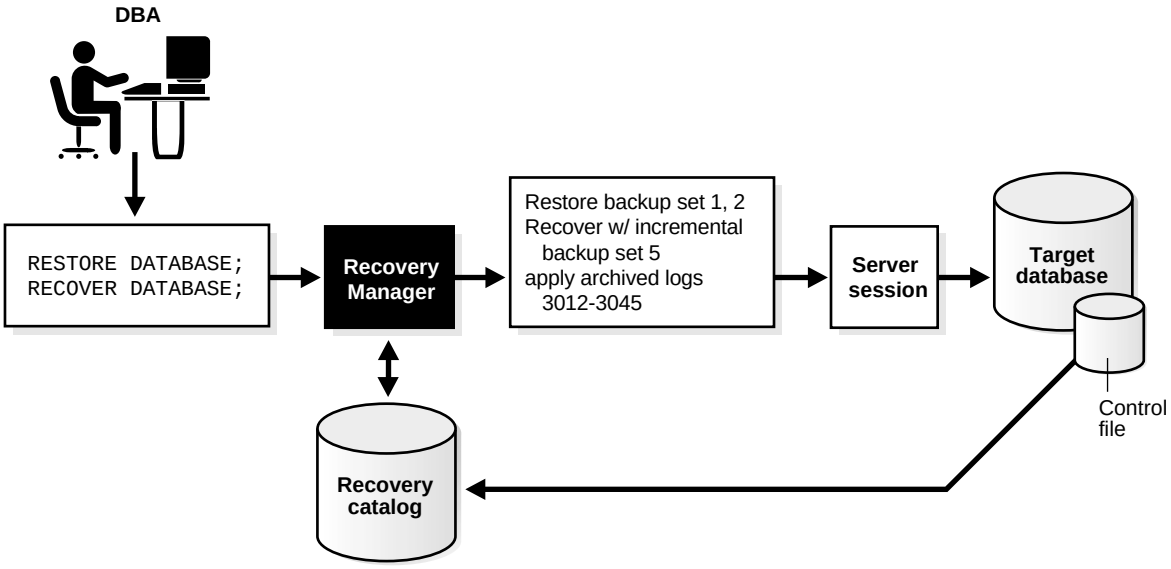
1. Place the database in the appropriate state: mounted or open. For example, mount the database when performing whole database recovery, or open the database when performing online tablespace recovery.
2. To perform incomplete recovery, use the `SET UNTIL` command to specify the time, SCN, or log sequence number at which recovery terminates. Alternatively, specify the `UNTIL` clause on the `RESTORE` and `RECOVER` commands.
3. Restore the necessary files with the `RESTORE` command.
4. Recover the datafiles with the `RECOVER` command.
5. Place the database in its normal state. For example, open it or bring recovered tablespaces online.

Figure 3–1 illustrates an example of RMAN media recovery. The DBA runs the following commands:

```
RESTORE DATABASE;
RECOVER DATABASE;
```

RMAN then queries the repository, which in this example is a recovery catalog. The recovery catalog obtains its metadata from the target database control file. RMAN then decides which backup sets to restore, and which incremental backups and archived logs to use for recovery. A server session on the target database instance performs the actual work of restore and recovery.

Figure 3–1 Performing RMAN Media Recovery



See Also:

- Chapter 8, "Advanced RMAN Recovery Techniques" for detailed restore and recovery procedures
- "Control File and Server Parameter File Autobackups" on page 2-38
- *Oracle Database Recovery Manager Reference* for RESTORE syntax
- *Oracle Database Recovery Manager Reference* for RECOVER syntax

Mechanics of Recovery: Incremental Backups and Redo Logs

If RMAN has a choice between applying an incremental backup or applying redo to the restored datafiles to meet a recovery objective, then it always chooses an incremental backup. If overlapping levels of incremental backup are available, then RMAN automatically chooses the one covering the longest period of time.

RMAN does not need to apply incremental backups to a restored level 0 incremental backup: it can also apply archived logs. RMAN restores the datafiles that it needs from available backups, applies incremental backups to the datafiles if they are available, and then applies archived logs.

How RMAN Searches for Archived Redo Logs During Recovery

If RMAN cannot find an incremental backup, then it looks in the repository for the names of archived redo logs to use for recovery. The database records an archived log in the control file whenever one of the following occurs:

- The archiver process archives a redo log
- RMAN restores an archived log
- The RMAN BACKUP AS COPY command copies a log
- The RMAN CATALOG command catalogs a user-managed backup of an archived log

If you use a recovery catalog, then RMAN propagates archived log data into the recovery catalog during resynchronization, classifying archived logs as image copies. You can view the log information through:

- The LIST ARCHIVELOG command
- The V$BACKUP_FILES control file view
- The V$ARCHIVED_LOG control file view

During recovery, RMAN looks for the needed logs using the filenames specified in the V$ARCHIVED_LOG view. If the logs were created in multiple destinations or were generated by the BACKUP AS COPY, CATALOG, or RESTORE commands, then multiple, identical copies of each log sequence number exist on disk. RMAN does not have a preference for one copy over another during recovery: all copies of a log sequence number listed as AVAILABLE are candidates. In a sense, RMAN is blind to the fact that the logs were generated in different destinations or in different ways.

If the RMAN repository indicates that a log has been deleted or uncataloged, then RMAN ceases to consider it as available for recovery. For example, assume that the database archives log 100 to directories /dest1 and /dest2. The RMAN repository indicates that /dest1/log100.arc and /dest2/log100.arc exist. If you delete /dest1/log100.arc with the DELETE command, then the repository indicates that only /dest2/log100.arc is available for recovery.

If the RMAN repository indicates that no copies of a needed log sequence number exist on disk, then RMAN looks in backups and restores archived redo logs as needed to perform the media recovery. By default, RMAN restores the archived redo logs to the first local archiving destination specified in the initialization parameter file. You can run the SET ARCHIVELOG DESTINATION command to specify a different restore location. If you specify the DELETE ARCHIVELOG option on RECOVER, then RMAN deletes the archived logs after restoring and applying

them. If you also specify MAXSIZE *integer* on the RECOVER command, then RMAN restores archived logs until the disk space allowed by MAXSIZE is consumed, then applies redo from the logs and deletes the restored logs to free space, until there is room enough to restore another archived log. RMAN continues restoring, applying and deleting logs, within the MAXSIZE limit, until recovery is complete.

RMAN Behavior When the Repository Is Not Synchronized

If an archived log is deleted from disk and the repository does not reflect this fact, then RMAN does not perform automatic failover during recovery. For example, if the repository indicates that /dest1/log100.arc is on disk when in fact this log was deleted using an operating system command, and if RMAN attempts to apply this log file during recovery, then recovery terminates with an error. RMAN does not automatically attempt to apply other copies of log 100 that are listed as available in the repository.

This situation can sometimes occur when you delete an archived log with an operating system utility and then fail to run CROSSCHECK to synchronize the repository. If you run a CROSSCHECK so that the repository is synchronized, then recovery can proceed by applying available copies of the log or restoring a backup of the log if no disk copies are available.

> **See Also:** *Oracle Database Recovery Manager Reference* for CROSSCHECK syntax

Incomplete Recovery

RMAN can perform either complete or incomplete recovery. You can specify a time, SCN, or log sequence number as a limit for incomplete recovery with the SET UNTIL command or with an UNTIL clause specified directory on the RESTORE and RECOVER commands. The easiest method is run the SET UNTIL command before issuing the RESTORE and RECOVER commands. After performing incomplete recovery, you must open the database with the RESETLOGS option.

> **See Also:** *Oracle Database Recovery Manager Reference* for the UNTIL clause syntax

Tablespace Point-in-Time Recovery

Recovery Manager automated Tablespace Point-in-Time Recovery (TSPITR) enables you to recover one or more tablespaces to a point in time that is different from that of the rest of the database. RMAN TSPITR is most useful in these cases:

- To recover from an erroneous drop or truncate table operation
- To recover a table that has become logically corrupted
- To recover from an incorrect batch job or other DML statement that has affected only a subset of the database
- In cases where there are multiple logical schemas in separate tablespaces of one physical database, and where one schema must be recovered to a point different from that of the rest of the physical database
- For VLDBs (very large databases), even if a full database point-in-time recovery would suffice, you might choose to do tablespace point-in-time recovery rather than restore the whole database from a backup and perform a complete database roll forward

Similar to a table export, RMAN TSPITR enables you to recover a consistent data set; however, the data set is the entire tablespace rather than a single object.

> **See Also:** Chapter 10, "RMAN Tablespace Point-in-Time Recovery (TSPITR)" to learn how to perform TSPITR using RMAN

Block Media Recovery with RMAN

Although datafile media recovery is the principal form of recovery, you can also use the RMAN BLOCKRECOVER command to perform **block media recovery**. Block media recovery recovers an individual corrupt datablock or set of datablocks within a datafile. In cases when a small number of blocks require media recovery, you can selectively restore and recover damaged blocks rather than whole datafiles.

Block media recovery provides several advantages over datafile media recovery. For example, block media recovery

- Lowers the **Mean Time to Recovery (MTTR)** because only blocks needing recovery are restored and only necessary corrupt blocks undergo recovery. Block media recovery minimizes redo application time and avoids I/O overhead during recovery.
- Allows affected datafiles to remain online during recovery of the blocks. Without block-level recovery, if even a single block is corrupt, then you must

restore a backup of the entire datafile and apply all redo generated for that file after the backup was created.

Note these restrictions of block media recovery:

- You can only perform block media recovery with RMAN. No SQL*Plus recovery interface is available.
- You can only perform complete recovery of individual blocks. In other words, you cannot stop recovery before all redo has been applied to the block.
- You can only recover blocks marked media corrupt. The `V$DATABASE_BLOCK_CORRUPTION` view indicates which blocks in a file were marked corrupt since the most recent `BACKUP` or `BACKUP ... VALIDATE` command was run against the file.
- You must have a full RMAN backup. Incremental backups are not allowed. Note that Block media recovery is able to restore blocks from parent incarnation backups and recover the corrupted blocks through a `RESETLOGS`.
- Blocks that are marked media corrupt are not accessible to users until recovery is complete. Any attempt to use a block undergoing media recovery results in an error message indicating that the block is media corrupt.

> **See Also:** "Performing Block Media Recovery with RMAN" on page 8-21 and *Oracle Database Recovery Manager Reference* for `BLOCKRECOVER` syntax

When Block Media Recovery Should Be Used

Block media recovery is not intended for cases where the extent of data loss or corruption is unknown and the entire datafile requires recovery. In such cases, datafile media recovery is the best solution. Block media recovery is not a replacement for traditional datafile media recovery, but a supplement to it.

In most cases, the database marks a block as media corrupt, invalidates the block in the instances (or all enabled instances in a Real Application Clusters configuration), and then writes it to disk when the corruption is first encountered. No subsequent read of the block will be successful until the block is recovered. You can only perform block recovery on blocks that are marked corrupt. This corrupt status effectively takes the block offline in all database instances and prevents user access during recovery.

Block media recovery is most useful for data losses that affect specific blocks. Block-level data loss usually results from intermittent, random I/O errors that do

not cause widespread data loss, as well as memory corruptions that get written to disk. Typically, these types of block corruption are reported in the following locations:

- Error messages in standard output
- The alert log
- User trace files
- Results of the SQL commands ANALYZE TABLE and ANALYZE INDEX
- Results of the DBVERIFY utility
- Third-party media management output

For example, you may discover the following messages in a user trace file:

```
ORA-01578: ORACLE data block corrupted (file # 7, block # 3)
ORA-01110: data file 7: '/oracle/oradata/trgt/tools01.dbf'
ORA-01578: ORACLE data block corrupted (file # 2, block # 235)
ORA-01110: data file 2: '/oracle/oradata/trgt/undotbs01.dbf'
```

You can then specify the corrupt blocks in the BLOCKRECOVER command as follows:

```
BLOCKRECOVER
  DATAFILE 7 BLOCK 3
  DATAFILE 2 BLOCK 235;
```

Block Media Recovery When Redo Is Missing

Like datafile media recovery, block media recovery cannot generally survive a missing or inaccessible archived log (although it will attempt restore failover when looking for usable copies of archived redo log files, as described in "Restore Failover" on page 3-4). Nevertheless, block media recovery can survive gaps in the redo stream if the missing or corrupt redo records do not affect the blocks being recovered. Whereas datafile recovery requires an unbroken series of redo changes from the beginning of recovery to the end, block media recovery only requires an unbroken set of redo changes for the blocks being recovered.

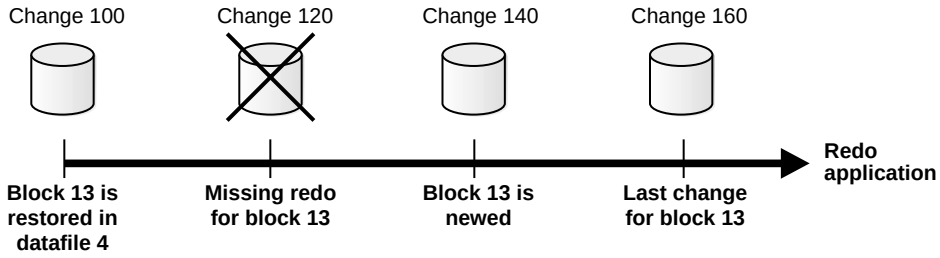
> **Note:** Each block is recovered independently during block media recovery, so recovery may be successful for a subset of blocks.

When RMAN first detects missing or corrupt redo records during block media recovery, it does not immediately signal an error because the block undergoing

recovery may become a **newed block** later in the redo stream. When a block is newed all previous redo for that block becomes irrelevant because the redo applies to an old incarnation of the block. For example, the database can new a block when users delete all the rows recorded in the block or drop a table.

Assume that media recovery is performed on block 13 as depicted in Figure 3–2.

Figure 3–2 Performing RMAN Media Recovery



After block recovery begins, RMAN discovers that change 120 is missing. RMAN does not terminate recovery in the hope that block 13 will be newed later in the redo stream. Assume that in change 140 a user drops the table `EMPLOYEE` stored in block 13. At this point, the database formats block 13 as a new block. Because the redo for block 13 in change 120 related to the `EMPLOYEE` table, and the `EMPLOYEE` table was dropped in change 140, RMAN can skip this missing change and apply the redo between changes 140 and 160.

Database Duplication with RMAN

Use the RMAN `DUPLICATE` command to create a copy of the target database in another location. The command restores backups of the primary database files and creates a new database.

As part of the duplication, RMAN manages the following:

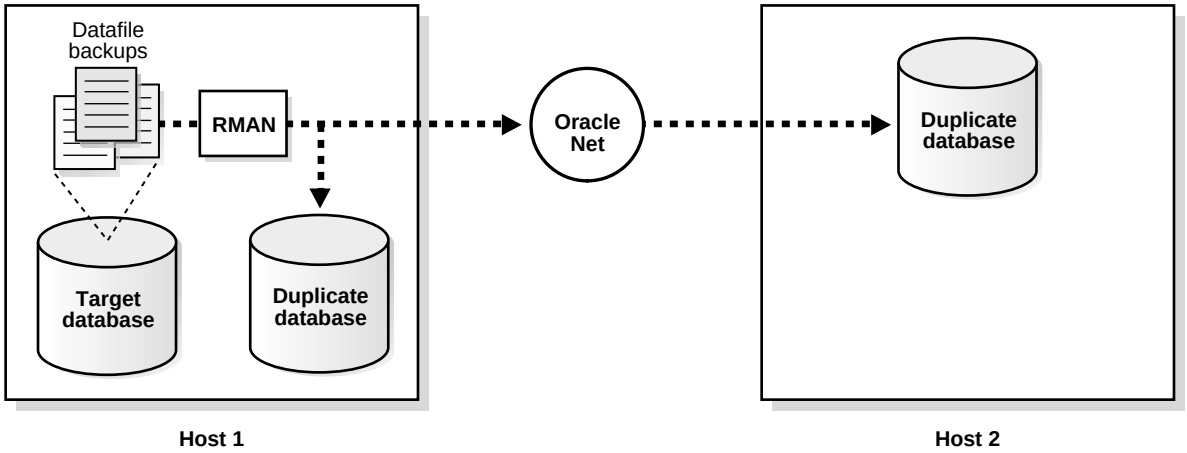
- Restores the target datafiles into the duplicate database and performs incomplete recovery using all available archived log and incremental backups
- Opens the duplicate database with the `RESETLOGS` option after incomplete recovery to create the online redo logs
- Generates a new, unique database identifier for the duplicate database

Note the following features of RMAN duplication. You can:

- Skip read-only tablespaces with the SKIP READONLY clause (read-only tablespaces are included by default). You can also exclude any tablespace with the SKIP TABLESPACE clause so long as it is not the SYSTEM or SYSAUX tablespace and does not contain rollback or undo data. If you omit tablespaces, then you can add them later.
- Create the duplicate database in a new host. If the same directory structure is available, then you can use the NOFILENAMECHECK option and reuse the target datafile filenames for the duplicate datafiles.
- Create the duplicate database by using the SET UNTIL command or UNTIL clause of the DUPLICATE command to recover it to a past time. By default, the DUPLICATE command creates the database using the most recent backups of the target database and then performs recovery to the most recent consistent point contained in the incremental and archived redo log backups.
- Use the duplicate database without a recovery catalog.
- Register the duplicate database in the same recovery catalog as the target database. This option is possible because the duplicate database receives a new database identifier during duplication. If you copy the target database with operating system utilities, then the database identifier of the copied database remains the same so you cannot register it in the same recovery catalog (unless you change its DBID with the DBNEWID utility, described in *Oracle Database Utilities*).

Figure 3–3 illustrates a case of database duplication. In this example, RMAN creates two duplicate database by using one set of datafile backups: one database on the local host and one database on a remote host.

Figure 3–3 Creating a Duplicate Database from Backups



The method you use to duplicate your database depends on whether you are creating your duplicate database on the same or a different host and whether the duplicate directory structure is the same as your target database directory structure. For example, in some cases you can keep the same directory structure and filenames in your duplicate database, while other times you must rename the files.

See Also:

Chapter 11, "Duplicating a Database with Recovery Manager" to learn how to make a duplicate database

Oracle Database Recovery Manager Reference for `DUPLICATE` command syntax

Oracle Database Utilities to learn how to use the `DBNEWID` utility

Physical Standby Database Creation with RMAN

You can use the RMAN `DUPLICATE` command to create a physical standby database. (Note that RMAN cannot be used to create a logical standby database, because it is not a block-for-block duplicate of the primary database.) RMAN automates the following steps of the creation procedure:

1. Restores the standby control file.
2. Restores the primary datafile backups.

3. Optionally, RMAN recovers the standby database (after the control file has been mounted) up to the specified time or to the latest archived redo log generated.
4. RMAN leaves the database mounted so that the user can activate it, place it in manual or managed recovery mode, or open it in read-only mode.

RMAN cannot fully automate creation of the standby database because you must manually create an initialization parameter file for the standby database, start the standby instance without mounting the control file, and perform any Oracle Net setup required before performing the creation of the standby. Also, you must have RMAN backups of all datafiles available as well as a control file backup that is usable as a standby control file.

RMAN can back up the standby datafiles, control file and archived redo logs of a physical standby database. Backups of datafiles and archived redo logs taken from a physical standby database are fully interchangeable with primary backups. In other words, you can restore a backup of a physical standby datafile to the primary database, and you can restore a backup of a primary datafile to the physical standby database. The standby control file backups can be used to restore the standby control file without needing to re-instantiate the standby in cases where the standby control file is lost.

> **See Also:** *Oracle Data Guard Concepts and Administration* to learn how to create and back up a physical standby database with RMAN

4

RMAN Maintenance Concepts

This chapter describes the basic concepts involved in using the Recovery Manager (RMAN) utility.

This chapter contains these topics:

RMAN Reporting

The RMAN repository contains extensive records of about backups as well as other useful information such as database schema and configuration settings. You can use RMAN commands LIST, REPORT, and SHOW to access this repository information.

In addition to these general reporting commands, you can also make use of the RESTORE... PREVIEW command to see which backup files are required to restore specific database objects from backup. See *Oracle Database Backup and Recovery Basics* for more details on RESTORE... PREVIEW.

Using the RMAN LIST Command

The LIST command is used to query the RMAN repository and obtain data about:

- Backup sets and image copies generated by the RMAN BACKUP command;
- Specified objects contained in the BACKUP-generated files, that is, archived logs, datafiles, control files, and server parameter files;
- Incarnations of a specified database, or of all databases known to a recovery catalog.

RMAN LIST output is sent either to standard output or to the message log (though not to both at the same time). You can also control how the output is organized as well as the level of detail in the output.

You can also list backups by querying V$BACKUP_FILES and the RC_BACKUP_FILES recovery catalog view. These views provide access to the same information as the LIST BACKUPSET command.

The LIST command displays the same files that the CROSSCHECK and DELETE commands operate on. Consequently, you can issue LIST to see what is in the repository, and then run CROSSCHECK to ensure that these files exist on disk or tape.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to generate lists
- "Querying the Recovery Catalog Views" on page 13-29 to learn how to use views as an alternative to `LIST`
- *Oracle Database Recovery Manager Reference* for `LIST` command syntax
- *Oracle Database Recovery Manager Reference* for `LOG` command-line syntax

RMAN Reports

RMAN reports are intended to provide analysis of your backup and recovery situation. An RMAN report can answer questions such as:

- Which datafiles need a backup?
- Which backups are obsolete because they are redundant or because they are not needed for recovery within a recovery window?
- Are any datafiles now unrecoverable because they have been the target of unrecoverable operations?
- What is the current physical schema of the database, or what was it at some previous time?
- Which backups are **orphaned**, that is, unusable in a restore operation, because they belong to incarnations of the database that are not direct predecessors of the current incarnation?

RMAN's reporting can be used to monitor and validate your ongoing backup strategy. The `REPORT NEED BACKUP` and `REPORT UNRECOVERABLE` commands let you ensure that the necessary backups are available for media recovery, and that you can perform media recovery within a reasonable amount of time.

Also, if you are managing backup storage yourself instead of using a flash recovery area, then you should run `REPORT OBSOLETE` regularly to identify backups no longer needed to meet your retention policy. You can then delete these backups with `DELETE OBSOLETE`.

Note: A datafile that does not have a backup is still considered recoverable by RMAN, as long as a complete set of archived redo logs is available, from the time the datafile was created to the present. During recovery, an empty datafile is created, and then all of the changes to the datafile from the archived redo logs are applied to reconstruct the full contents of the file.

Reports of Obsolete Backups

The REPORT OBSOLETE command displays backups of datafiles, control files, and archived redo logs that can be deleted because they are no longer needed. You can define what makes a file obsolete in the following mutually exclusive ways:

Parameter	Meaning
REDUNDANCY *integer*	At least *integer* more recent backups of this file already exist.
RECOVERY WINDOW *integer*	The backup is not needed for recovery to any point within the recovery window of *integer* days. For each datafile, one backup that is older than the recovery window must exist. In other words, one backup of each datafile must satisfy the condition SYSDATE - CHECKPOINT_TIME >= RECOVERY WINDOW. All backups older than the most recent backup that satisfies this condition are obsolete.

In addition to obsolete datafile backups, RMAN reports obsolete archived logs and archived log backups. Regardless of which parameter is specified, RMAN uses this setting to determine which backups of datafiles are no longer needed, which in turn determines when archived logs (and backups of archived logs) are no longer needed. Note that if a datafile has never been backed up, then all archived redo logs back to the creation time of the file will be retained. With a full set of logs, the file can be completely re-created during media recovery. An empty datafile is automatically created during recovery, and all changes ever applied to the original datafile that was not backed up are re-applied to the newly created file.

The REPORT OBSOLETE command lets you identify files which are no longer needed to satisfy backup retention policies. By default, the REPORT OBSOLETE commannd reports which files are obsolete under the currently configured retention policy. To generate reports of which files are obsolete according to different retention policies by using REDUNDANCY or RECOVERY WINDOW retention policy options with the REPORT OBSOLETE command. For example, if you run any of these commands:

```
RMAN> REPORT OBSOLETE REDUNDANCY 2;
RMAN> REPORT OBSOLETE RECOVERY WINDOW OF 5 DAYS;
```

RMAN displays backups that are obsolete according to those retention policies, regardless of the actual configured retention policy.

If you disable the retention policy completely (that is, if you run CONFIGURE RETENTION POLICY TO NONE), then RMAN does not consider any backups to be obsolete. If you run REPORT OBSOLETE with no options and no retention policy is configured, then RMAN issues an error message.

You can also query V$BACKUP_FILES and RC_BACKUP_FILES, using the OBSOLETE column to identify backup sets, datafile copies, and archived logs that are obsolete according to the configured retention policy.

Note: An obsolete backup differs from an expired backup. An obsolete backup is no longer needed according to the user's retention policy. An expired backup is a backup that the CROSSCHECK command fails to find on the specified media device.

See Also: *Oracle Database Recovery Manager Reference* for CONFIGURE command syntax

Reports of Orphaned Backups

The REPORT OBSOLETE ORPHAN command displays **orphaned backups**. To understand orphaned backups, you must understand the ide aof a database **incarnation**.

Understanding Database Incarnations A new incarnation of a database is created whenever each time the database is opened with the RESETLOGS option. Performing an OPEN RESETLOGS archives the current online redo logs, resets the log sequence number to 1, and then gives the online redo logs a new time stamp and SCN.

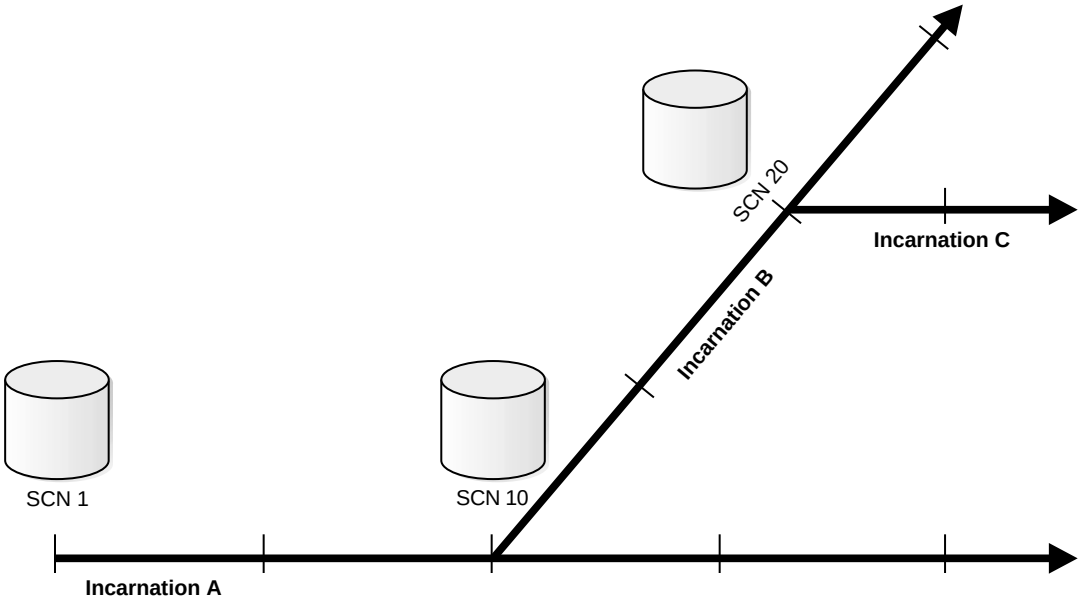
RMAN is able to restore backups from direct ancestor incarnations and recover to the current time, even across OPEN RESETLOGS operations, as long as a continuous path of archived logs exists from the earliest backups to the point to which you want to recover.

When a database goes through multiple incarnations, some backups can become orphaned. Orphaned backups are backups that are unusable because they belong to

incarnations of the database that are not direct ancestors of the current incarnation. That is, they are not in an unbroken incarnation path from the current incarnation.

Figure shows a database that goes through three incarnations.

Figure 4–1 Database Incarnations and Orphaned Backups



Incarnation A of the database started at SCN 1. At SCN 10, assume that you performed a RESETLOGS operation and created incarnation B. At SCN 20, you performed another RESETLOGS operation on incarnation B and created a new incarnation C.

The following table explains which backups in this example are orphans, depending on which incarnation is current.

Current Incarnation	Usable Backups (Nonorphaned)	Orphaned Backups
Incarnation A	All backups from incarnation A	All backups from incarnations B and C

Current Incarnation	Usable Backups (Nonorphaned)	Orphaned Backups
Incarnation B	■ All backups from incarnation A prior to SCN 10 ■ All backups from incarnation B	■ Backups from incarnation A after SCN 10. ■ All backups from incarnation C
Incarnation C	■ All backups from incarnation A prior to SCN 10 ■ All backups from incarnation B prior to SCN 20 ■ All backups from incarnation C	■ All backups from incarnation A after SCN 10 ■ All backups from incarnation B after SCN 20

See Also: *Oracle Database Backup and Recovery Basics* to learn how to generate reports, and *Oracle Database Recovery Manager Reference* for REPORT syntax

SHOW Command Output

The SHOW command can display any configuration set by the CONFIGURE command. For example, to display the CONFIGURE CHANNEL settings, run SHOW CHANNEL. You can run SHOW ALL to display all current configurations. This configuration data is also stored in the V$RMAN_CONFIGURATION view.

See Also: *Oracle Database Recovery Manager Reference* for SHOW syntax

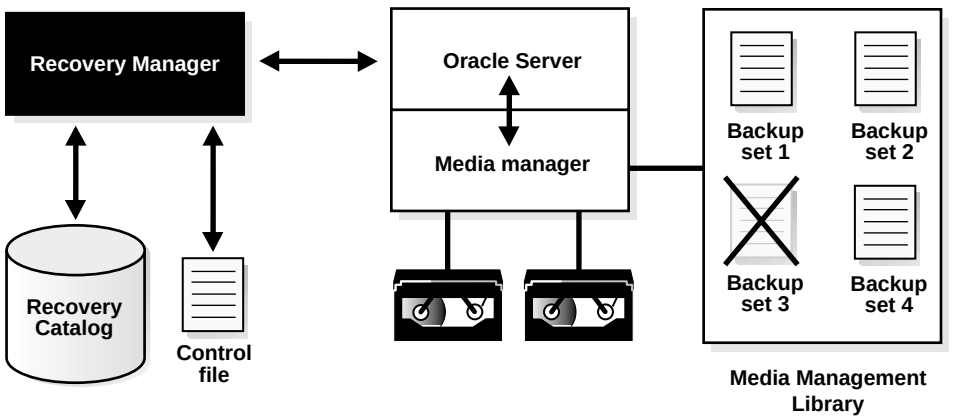
Crosschecks of RMAN Backups

RMAN's record of backups can become out of step with the actual backups that exist on tape or disk. For example, a user may inadvertently delete backup pieces from disk using operating system commands, or one of the tapes used by the media manager may become corrupted.

To ensure that data about backups in the recovery catalog or control file is synchronized with actual files on disk or in the media management catalog, perform a **crosscheck**. The CROSSCHECK command operates only on files that are recorded in the RMAN repository.

Figure 4–2 illustrates a crosscheck of the media manager. RMAN queries the RMAN repository for the names and locations of the four backup pieces to be checked. RMAN sends this information to the target database server, which queries the media management software about the backups. The media management software then checks its media catalog and reports back to the server that backup set 3 is missing. RMAN updates the status of backup set 3 to EXPIRED in the repository. The record for backup set 3 will now be deleted if you run DELETE EXPIRED.

Figure 4–2 Crosschecking the Media Manager



Crosschecks are useful because they can

- Update outdated information about backups that disappeared from disk or tape or became corrupted
- Update the repository if you delete archived redo logs or other files with operating system commands

Use the crosscheck feature to check the status of a backup on disk or tape. If the backup is on disk, then CROSSCHECK checks whether the header of the file is valid. If a backup is on tape, then the command checks that the backups exist in the media management software's catalog.

Backup pieces and image copies can have the status AVAILABLE, EXPIRED, or UNAVAILABLE. You can view the status information in the output of the LIST command and the recovery catalog views.

You can issue the DELETE EXPIRED command to delete all expired backups. RMAN removes the record for the expired file from the repository. If for some

reason the file still exists on the media, then RMAN issues warnings and lists the mismatched objects that cannot be deleted.

Note: The CROSSCHECK command does *not* delete operating system files or remove repository records. You must use the DELETE command for these operations.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to perform crosschecks
- *Oracle Database Recovery Manager Reference* for CROSSCHECK syntax and a description of the possible status values
- *Oracle Database Recovery Manager Reference* for DELETE syntax

Monitoring RMAN Through V$ Views

When LIST, REPORT and SHOW do not provide all the information you need on RMAN activities, there are a number of useful V$ views that can provide more details.

Sometimes it is useful to identify exactly what a server session performing a backup or recovery task is doing. You have access to several views that can assist in monitoring the progress of or obtaining information about RMAN jobs, as described in the following table.

View	Description
V$RMAN_OUTPUT	Displays messages reported by an RMAN job in progress.
V$RMAN_STATUS	Shows the success / failure status of all recently completed RMAN jobs.
V$PROCESS	Identifies currently active processes.
V$RECOVER_FILE	Identifies which datafiles require recovery.
V$SESSION	Identifies currently active sessions. Use this view to determine which database server sessions correspond to which RMAN allocated channels.
V$SESSION_LONGOPS	Provides progress reports on RMAN backup and restore jobs.

View	Description
V$SESSION_WAIT	Lists the events or resources for which sessions are waiting.
V$BACKUP_SYNC_IO	Displays rows when the I/O is synchronous to the process (or thread on some platforms) performing the backup.
V$BACKUP_ASYNC_IO	Displays rows when the I/O is asynchronous to the process (or thread on some platforms) performing the backup.

Asynchronous I/O is obtained either through the use of slave I/O processes or because it is supported by the underlying operating system.

You can use RMAN to perform the checks discussed in the following sections:

- Correlating Server Sessions with RMAN Channels
- Monitoring RMAN Job Progress
- Monitoring RMAN Interaction with the Media Manager
- Monitoring RMAN Job Performance
- Determining Which Datafiles Require Recovery

Correlating Server Sessions with RMAN Channels

To identify which server sessions correspond to which RMAN channels, you can query V$SESSION and V$PROCESS. The SPID column of V$PROCESS identifies the operating system ID number for the process or thread. For example, on UNIX the SPID column shows the process ID, whereas on Windows the SPID column shows the thread ID. You have two basic methods for obtaining this information, depending on whether you have multiple RMAN sessions active concurrently.

Matching Server Sessions with Channels When One RMAN Session Is Active

When only one RMAN session is active, the easiest method for determining the server session ID for an RMAN channel is to execute the following query on the target database while the RMAN job is executing:

```
COLUMN CLIENT_INFO FORMAT a30
COLUMN SID FORMAT 999
COLUMN SPID FORMAT 9999

SELECT s.SID, p.SPID, s.CLIENT_INFO
FROM V$PROCESS p, V$SESSION s
WHERE p.ADDR = s.PADDR
```

```
AND CLIENT_INFO LIKE 'rman%'
;
```

If you do not run the SET COMMAND ID command in the RMAN job, then the CLIENT_INFO column displays in the following format:

```
rman channel=channel_id
```

For example, the following shows sample output:

```
 SID SPID         CLIENT_INFO
---- ------------ ------------------------------
  14 8374         rman channel=ORA_SBT_TAPE_1
```

Matching Server Sessions with Channels in Multiple RMAN Sessions

If more than one RMAN session is active, it is possible for the V$SESSION.CLIENT_INFO column to yield the same information for a channel in each session. For example:

```
 SID SPID         CLIENT_INFO
---- ------------ ------------------------------
  14 8374         rman channel=ORA_SBT_TAPE_1
   9 8642         rman channel=ORA_SBT_TAPE_1
```

In this case, you have the following methods for determining which channel corresponds to which SID value.

Obtaining the Channel ID from the RMAN Output In this method, you must first obtain the sid values from the RMAN output and then use these values in your SQL query.

To correlate a process with a channel during a backup:

1. In one of the active sessions, run the RMAN job as normal and examine the output to get the sid for the channel. For example, the output may show:

   ```
   Starting backup at 21-AUG-01
   allocated channel: ORA_SBT_TAPE_1
   channel ORA_SBT_TAPE_1: sid=14 devtype=SBT_TAPE
   ```

2. Start a SQL*Plus session and then query the joined V$SESSION and V$PROCESS views *while the RMAN job is executing*. For example, enter:

   ```
   COLUMN CLIENT_INFO FORMAT a30
   COLUMN SID FORMAT 999
   COLUMN SPID FORMAT 9999
   ```

```
SELECT s.SID, p.SPID, s.CLIENT_INFO
FROM V$PROCESS p, V$SESSION s
WHERE p.ADDR = s.PADDR
AND CLIENT_INFO LIKE 'rman%'
/
```

Use the `sid` value obtained from the first step to determine which channel corresponds to which server session:

```
       SID SPID         CLIENT_INFO
---------- ------------ ------------------------------
        14 2036         rman channel=ORA_SBT_TAPE_1
        12 2066         rman channel=ORA_SBT_TAPE_1
```

Correlating Server Sessions with Channels by Using SET COMMAND ID In this method, you specify a command ID string in the RMAN backup script. You can then query `V$SESSION.CLIENT_INFO` for this string.

To correlate a process with a channel during a backup:

1. In each session, set the `COMMAND ID` to a different value *after* allocating the channels and then back up the desired object. For example, enter the following in session 1:

   ```
   RMAN> RUN
   {
     ALLOCATE CHANNEL c1 TYPE sbt;
     SET COMMAND ID TO 'sess1';
     BACKUP DATABASE;
   }
   ```

   Set the command ID to a string such as `sess2` in the job running in session 2:

   ```
   RUN
   {
     ALLOCATE CHANNEL c1 TYPE sbt;
     SET COMMAND ID TO 'sess2';
     BACKUP DATABASE;
   }
   ```

2. Start a SQL*Plus session and then query the joined `V$SESSION` and `V$PROCESS` views *while the RMAN job is executing*. For example, enter:

   ```
   SQL> SELECT SID, SPID, CLIENT_INFO
     FROM V$PROCESS p, V$SESSION s
     WHERE p.ADDR = s.PADDR
   ```

```
  AND CLIENT_INFO LIKE '%id=sess%';
```

If you run the SET COMMAND ID command in the RMAN job, then the CLIENT_INFO column displays in the following format:

```
id=command_id,rman channel=channel_id
```

For example, the following shows sample output:

```
 SID SPID         CLIENT_INFO
---- ------------ ------------------------------
  11 8358         id=sess1
  15 8638         id=sess2
  14 8374         id=sess1,rman channel=c1
   9 8642         id=sess2,rman channel=c1
```

The rows that contain the string rman channel show the channel performing the backup. The remaining rows are for the connections to the target database.

> **See Also:** *Oracle Database Recovery Manager Reference* for SET COMMAND ID syntax, and *Oracle Database Reference* for more information on V$SESSION and V$PROCESS

Monitoring RMAN Job Progress

Monitor the progress of backups and restores by querying the view V$SESSION_LONGOPS. RMAN uses two types of rows in V$SESSION_LONGOPS: detail and aggregate rows. Detail rows describe the files being processed by one job step, while aggregate rows describe the files processed by all job steps in an RMAN command. A job step is the creation or restore of one backup set or datafile copy. Detail rows are updated with every buffer that is read or written during the backup step, so their granularity of update is small. Aggregate rows are updated when each job step completes, so their granularity of update is large.

Table 4–1 describes column in V$SESSION_LONGOPS that are most relevant for RMAN. Typically, you will view the detail rows rather than the aggregate rows to determine the progress of each backup set.

Table 4–1 Columns of V$SESSION_LONGOPS Relevant for RMAN

Column	Description for Detail Rows
SID	The server session ID corresponding to an RMAN channel.
SERIAL#	The server session serial number. This value changes each time a server session is reused.

Table 4–1 Columns of V$SESSION_LONGOPS Relevant for RMAN

Column	Description for Detail Rows
OPNAME	A text description of the row. Examples of details rows include `RMAN: datafile copy`, `RMAN: full datafile backup`, and `RMAN: full datafile restore`. **Note:** `RMAN: aggregate input` and `RMAN: aggregate output` are the only aggregate rows.
CONTEXT	For backup output rows, this value is `2`. For all other rows except proxy copy (which does not update this column), the value is `1`.
SOFAR	The meaning of this column depends on the type of operation described by this row: ■ For image copies, the number of blocks that have been read. ■ For backup input rows, the number of blocks that have been read from the files being backed up. ■ For backup output rows, the number of blocks that have been written to the backup piece. ■ For restores, the number of blocks that have been processed to the files that are being restored in this one job step. ■ For proxy copies, the number of files that have been copied.
TOTALWORK	The meaning of this column depends on the type of operation described by this row: ■ For image copies, the total number of blocks in the file. ■ For backup input rows, the total number of blocks to be read from all files processed in this job step. ■ For backup output rows, the value is `0` because RMAN does not know how many blocks that it will write into any backup piece. ■ For restores, the total number of blocks in all files restored in this job step. ■ For proxy copies, the total number of files to be copied in this job step.

Each server session performing a backup or restore reports its progress compared to the total amount of work required for a job step. For example, if you perform a database restore that uses two channels, and each channel has two backup sets to restore (a total of four sets), then each server session reports its progress through a single backup set. When that set is completely restored, RMAN begins reporting progress on the next set to restore.

To monitor job progress:

1. Before starting the job, create a script file (called, for this example, `longops`) containing the following SQL statement:

```
SELECT SID, SERIAL#, CONTEXT, SOFAR, TOTALWORK,
       ROUND(SOFAR/TOTALWORK*100,2) "%_COMPLETE"
FROM V$SESSION_LONGOPS
WHERE OPNAME LIKE 'RMAN%'
  AND OPNAME NOT LIKE '%aggregate%'
  AND TOTALWORK != 0
  AND SOFAR <> TOTALWORK
;
```

2. After connecting to the target database and, if desired, the recovery catalog database, start an RMAN job. For example, enter:

```
RESTORE DATABASE;
```

3. While the job is running, start SQL*Plus connected to the target database, and execute the `longops` script to check the progress of the RMAN job. If you repeat the query while the restore progresses, then you see output such as the following:

```
SQL> @longops
       SID    SERIAL#    CONTEXT      SOFAR  TOTALWORK %_COMPLETE
---------- ---------- ---------- ---------- ---------- ----------
         8         19          1      10377      36617      28.34

SQL> @longops
       SID    SERIAL#    CONTEXT      SOFAR  TOTALWORK % COMPLETE
---------- ---------- ---------- ---------- ---------- ----------
         8         19          1      21513      36617      58.75

SQL> @longops
       SID    SERIAL#    CONTEXT      SOFAR  TOTALWORK % COMPLETE
---------- ---------- ---------- ---------- ---------- ----------
         8         19          1      29641      36617      80.95

SQL> @longops
       SID    SERIAL#    CONTEXT      SOFAR  TOTALWORK % COMPLETE
---------- ---------- ---------- ---------- ---------- ----------
         8         19          1      35849      36617       97.9

SQL> @longops
no rows selected
```

4. If you run the script at intervals of two minutes or more and the %_COMPLETE column does not increase, then RMAN is encountering a problem. Refer to "Monitoring RMAN Interaction with the Media Manager" on page 4-16 to obtain more information.

If you frequently monitor the execution of long-running tasks, you could create a shell script or batch file under your host operating system that runs SQL*Plus to execute this query repeatedly.

Monitoring RMAN Interaction with the Media Manager

You can use the event names in the dynamic performance event views to monitor RMAN calls to the media management API. The event names have one-to-one correspondence with sbt functions, as shown in the following examples:

```
sbtinit
sbtopen
sbtread
sbtwrite
sbtbackup
```

Before making a call to any of functions in the media management API, the server adds a row in V$SESSION_WAIT, with the STATUS column including the string WAIT. The V$SESSION_WAIT.SECONDS_IN_WAIT column shows the number of seconds that the server has been waiting for this call to return. After an sbt function is returned from the media manager, this row disappears.

A row in V$SESSION_WAIT corresponding to an sbt event name does not indicate a problem, because the server updates these rows at runtime. The rows appear and disappear as calls are made and returned. However, if the SECONDS_IN_WAIT column is high, then the media manager may be hung.

To monitor the sbt events, you can run the following SQL query:

```
COLUMN EVENT FORMAT a10
COLUMN SECONDS_IN_WAIT FORMAT 999
COLUMN STATE FORMAT a20
COLUMN CLIENT_INFO FORMAT a30

SELECT p.SPID, EVENT, SECONDS_IN_WAIT AS SEC_WAIT,
       STATE, CLIENT_INFO
FROM V$SESSION_WAIT sw, V$SESSION s, V$PROCESS p
WHERE sw.EVENT LIKE 'sbt%'
       AND s.SID=sw.SID
```

```
       AND s.PADDR=p.ADDR
/
```

Examine the SQL output to determine which `sbt` functions are waiting. For example, the following output indicates that RMAN has been waiting for the `sbtbackup` function to return for ten minutes:

```
SPID EVENT        SEC_WAIT STATE               CLIENT_INFO
---- ---------- ---------- ------------------- ------------------------------
8642 sbtbackup         600 WAITING             rman channel=ORA_SBT_TAPE_1
```

Note: The `V$SESSION_WAIT` view shows only database events, not media manager events.

See Also: *Oracle Database Reference* for descriptions of `V$SESSION_WAIT`

Monitoring RMAN Job Performance

Monitor backup and restore performance by querying `V$BACKUP_SYNC_IO` and `V$BACKUP_ASYNC_IO`.

See Also: *Oracle Database Reference* for more information on these V$ views, and "Step 5: Query V$ Views to Identify Bottlenecks" on page 14-11 to learn how to use these views to tune backup performance

Determining Which Datafiles Require Recovery

You can often use the dynamic performance view `V$RECOVER_FILE` to determine which files need to be recovered and why they need to be recovered. The following query shows the file numbers of datafiles that require recovery, as well as the reason for recovery (if known) and the SCN and time when recovery needs to begin:

```
COL FILE# FORMAT 999
COL ERROR FORMAT a10
SELECT * FROM V$RECOVER_FILE;

FILE# ONLINE  ONLINE_ ERROR         CHANGE# TIME
----- ------- ------- ---------- ---------- -------------------
    4 ONLINE  ONLINE  FILE NOT            0
                      FOUND
    5 ONLINE  ONLINE  FILE NOT            0
```

```
                    FOUND
  8 OFFLINE OFFLINE OFFLINE             0
                    NORMAL
```

> **Note:** The view is not useful if the control file currently in use is a restored backup or a new control file created after the media failure occurred. A restored or re-created control file does not contain the information needed to update V$RECOVER_FILE accurately.

You canperform a useful join between V$RECOVER_FILE, V$DATAFILE and V$TABLESPACE to see which datafiles and tablespaces are in need of recovery, as shown in the following example:

```
COL df# FORMAT 999
COL df_name FORMAT a35
COL tbsp_name FORMAT a10
COL status FORMAT a7
COL error FORMAT a10

SELECT r.FILE# AS df#, d.NAME AS df_name, t.NAME AS tbsp_name,
       d.STATUS, r.ERROR, r.CHANGE#, r.TIME
FROM V$RECOVER_FILE r, V$DATAFILE d, V$TABLESPACE t
WHERE t.TS# = d.TS#
AND d.FILE# = r.FILE#
;
```

Sample output follows:

```
 DF# DF_NAME                            TBSP_NAME  STATUS   ERROR         CHANGE# TIME
---- ---------------------------        ---------- -------  ---------- ---------- -----
   4 /oracle/oradata/trgt/drsys01.dbf   DRSYS      ONLINE   FILE NOT           0
                                                            FOUND
   5 /oracle/oradata/trgt/example01.dbf EXAMPLE    ONLINE   FILE NOT           0
                                                            FOUND
   8 /oracle/oradata/trgt/users01.dbf   USERS      OFFLINE  OFFLINE            0
                                                            NORMAL
```

Deletion of RMAN Backups

Every RMAN backup produces a corresponding record in the RMAN repository. This record is stored in the control file. If a recovery catalog is used, the record can

also be found in the recovery catalog after the recovery catalog is resynced from the control file.

For example, if you generate a full database backup set, then you can view the record for this backup set in the V$BACKUP_SET control file view. If you use a recovery catalog, then you can also access the record in the RC_BACKUP_SET catalog view.

The V$ control file views and recovery catalog tables differ in the way that they store information, and this affects how RMAN handles repository records. The recovery catalog RMAN repository is stored in actual database tables, while the control file version of the repository is stored in an internal structure in the control file.

When you use an RMAN command to delete a backup, RMAN performs the following steps:

- Removes the physical file from the operating system
- Updates the backup records in the control file to status DELETED
- Removes the backup records from the recovery catalog tables (if a recovery catalog is used)

Because of the way that control file data is stored, RMAN cannot remove the record from the control file, only update it to DELETED status. However, because the catalog tables are ordinary database tables, RMAN removes rows from them.

Summary of RMAN Deletion Methods

Table 4–2 describes the functionality of the various RMAN deletion commands. All of these work whether you store the RMAN repository only in the control file or use a recovery catalog.

Table 4–2 Maintenance Commands and Scripts **(Page 1 of 2)**

Command or Script	Purpose
DELETE	To delete physical backups, update the control file records to status DELETED, and remove their records from the recovery catalog (if a recovery catalog is used). You can specify that DELETE should remove backups that are EXPIRED or OBSOLETE. If you run DELETE EXPIRED on a backup that exists, RMAN issues a warning and does not delete the backup. You can override this behavior and delete the backup by running DELETE FORCE.

Table 4–2 Maintenance Commands and Scripts **(Page 2 of 2)**

Command or Script	Purpose
`BACKUP ... DELETE [ALL] INPUT`	To back up archived logs, datafile copies, or backup sets, then delete the input files from the operating system after the successful completion of the backup. RMAN also deletes and updates repository records for the deleted input files. If you specify `DELETE INPUT` (without `ALL`), then RMAN deletes only the specific files that it backs up. If you specify `ALL INPUT`, then RMAN deletes all copies of the files recorded in the RMAN repository.
`CHANGE ... UNCATALOG`	To delete recovery catalog records for specified backups and change their control file records to status `DELETED`. Note that the `CHANGE ... UNCATALOG` command does not delete files from the operating system.

See Also: "Crosschecks of RMAN Backups" on page 4-7

Removal of Backups with the DELETE Command

The `DELETE` command can remove any file that the `LIST` and `CROSSCHECK` commands can operate on. For example, you can delete backup sets, archived redo logs, and datafile copies. The `DELETE` command removes both the physical file and the catalog record for the file.

Advantage of Using DELETE Instead of Operating System Commands

Always use `DELETE` command within RMAN to remove RMAN backups, rather than an operating system or media manager utility or command. Otherwise, the RMAN repository can contain records of backups that are no longer available for use in restore operations.

If you delete backups without using RMAN, you can use one of the following methods within RMAN to update the RMAN repository directly without performing a crosscheck:

- Run `CROSSCHECK` to change the status of these files to `EXPIRED` and then run `DELETE EXPIRED` to delete the records from the RMAN repository
- Run `CHANGE ... UNCATALOG` to remove the catalog records

Deletion of Obsolete Backups

The DELETE OBSOLETE command provides a convenient way to delete backups that are no longer needed. It uses the same REDUNDANCY, RECOVERY WINDOW, and ORPHAN options as the REPORT OBSOLETE command.

If you have configured a retention policy, then you can run DELETE OBSOLETE periodically to delete all backups considered obsolete by this policy. For example, you can run DELETE OBSOLETE in a script every night with a scheduling utility, freeing disk and tape space used by backups that are no longer needed.

Note that using a flash recovery area as the destination for all backups eliminates the need to manage obsolete backups. Obsolete backups will be deleted from the flash recovery area automatically as disk space is needed to store backup-related files.

See Also: "Reports of Obsolete Backups" on page 4-4

Deletion of Expired Backups

The CROSSCHECK command updates the repository status for a backup to EXPIRED when it cannot locate it at the location to which it was backed up. This condition could occur if, for example, a backup was deleted from disk at the operating system level. You can identify expired backups by running the CROSSCHECK command as in the following example:

```
RMAN> CROSSCHECK BACKUP;

crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=0ad8d32i_1_1 recid=10 stamp=445025363
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=c-1334876723-20011105-00 recid=11 stamp=445025367
crosschecked backup piece: found to be 'EXPIRED'
backup piece handle=0cd8d361_1_1 recid=12 stamp=445025473
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=c-1334876723-20011105-01 recid=13 stamp=445025475
Crosschecked 4 objects
```

If you run CROSSCHECK while some backup device is temporarily not accessible. This can happen if a disk is unmounted or if RMAN does not correctly connect to a media manager. In such a case, fix the problem that prevented RMAN from finding the backups and rerun CROSSCHECK.

The DELETE EXPIRED command removes the recovery catalog records for expired backups, and updates their control file records to status DELETED.

This command is especially useful if a user inadvertently deletes RMAN backups or archived logs from disk with an operating system utility. In such a case, the RMAN repository is not synchronized with the actual contents of disk. By running the CROSSCHECK command, RMAN marks the backups that it cannot find as expired. Then, you can run DELETE EXPIRED to remove the records for these files.

Deletion of Archived Redo Logs That Are Already Backed Up

You may want to delete files such as archived logs only if they have been backed up a specified number of times to tape. The DELETE command supports this behavior. The following example deletes all archived redo logs that have already been backed up at least two times to tape:

```
RMAN> DELETE ARCHIVELOG ALL BACKED UP 2 TIMES TO DEVICE TYPE sbt;
```

Behavior of DELETE Command When the Repository and Media Do Not Correspond

The repository record for an object can sometimes fail to reflect the physical status of the object. For example, you backup an archived redo log to disk and then use an operating system utility to delete the object. If you do not run the CROSSCHECK command to update the repository, and if you then run DELETE against the object, then the repository shows that the object is AVAILABLE while the object is in fact missing. The following table indicates the behavior of DELETE in such situations.

Repository Status	Physical Status	Behavior of DELETE Command
AVAILABLE	Not found on media	Does not delete the object and reports the list of mismatched objects at the end of the job. RMAN does not update the repository status.
EXPIRED	Found on media	Does not delete the object and reports the list of mismatched objects at the end of the job. RMAN does not update the repository status.
UNAVAILABLE	Any	Removes repository record and deletes object if it exists. All I/O errors are ignored.

If you use the FORCE option of DELETE, RMAN will remove the repository record and delete the file if it exists. All I/O errors are ignored, and RMAN displays the number of objects deleted at the end of the job.

Removal of Backups with the BACKUP ... DELETE INPUT Command

The BACKUP . . . DELETE INPUT command can delete archived redo logs, datafile copies, and backup sets after backing them up. This functionality is especially useful when backing up archived logs on disk to tape. RMAN backs up one copy of each log sequence number, and then deletes the file that it backs up. For example, assume that you issue:

```
RMAN> BACKUP ARCHIVELOG ALL DELETE INPUT;
```

In this command, RMAN backs up one copy of each log for each available sequence number, and then deletes only the archived redo log file that it actually backs up. If you have multiple redo log archiving destinations, the other copies of the same log sequence number are not deleted.

If you specify the DELETE ALL INPUT option, then RMAN deletes whichever files match the criteria that you specify, even if there are several files of the same log sequence number. For example, assume that you archive to three different directories. Then, you issue this command:

```
RMAN> BACKUP ARCHIVELOG ALL FROM SEQUENCE 1200 DELETE ALL INPUT;
```

In this case, RMAN backs up only one copy of each log sequence between 1200 and the most recent sequence, but deletes all logs with these sequence numbers contained in the three archive destinations.

During backup of archived redo logs, RMAN checks the file being backed up for corruption. If corruption is found, RMAN automatically switches to reading another copy of the same archived redo log, if one exists. For example, assume that /log1 and /log2 are the only enabled archiving destinations, and that they contain logs with sequence number up through 150. You run this command:

```
RMAN> BACKUP ARCHIVELOG FROM SEQUENCE 123 DELETE ALL INPUT;
```

RMAN can start reading from any cpoy of a given log. For example, if RMAN starts reading the copy of log sequence 123 from /log1 and discovers corruption in the file, it continues reading from the copy in /log2. Because DELETE ALL INPUT is specified, RMAN deletes all copies of logs on disk of sequence 123 and higher.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to delete backups and exempt them from a retention policy
- *Oracle Database Recovery Manager Reference* for CHANGE syntax
- *Oracle Database Recovery Manager Reference* for DELETE syntax

CHANGE AVAILABLE and CHANGE UNAVAILABLE with RMAN Backups

RMAN can update the repository to show backups as AVAILABLE or UNAVAILABLE. An unavailable backup is one that cannot be accessed at a particular moment but that has not been deleted. For example, you may have backups on tape that are temporariliy stored offsite and are inaccesible. You can use the CHANGE . . . UNAVAILABLE command to update the repository status for these backups to UNAVAILABLE so that RMAN will not try to use them for its backup and recovery operations.

When the tapes become available again, you can issue the CHANGE . . . AVAILABLE command to update the RMAN repository to show that these backups now can be used. After setting the files back to status AVAILABLE, you can also run a CROSSCHECK to verify that RMAN can access the files.

See Also:

- *Oracle Database Backup and Recovery Basics* to learn how to make backups available or unavailable
- *Oracle Database Recovery Manager Reference* for CHANGE syntax

Changing Retention Policy Status of RMAN Backups

Use CHANGE... KEEP or CHANGE... NOKEEP to specify whether a backup should be subject to the configured retention policy or kept until a different date or even indefinitely.

The KEEP option exempts a backup from the current retention policy either indefinitely or until the specified UNTIL time. RMAN does not mark the files as obsolete even if they would be considered obsolete under the retention policy. Such backups are called **long-term backups**. CHANGE . . . NOKEEP is used to undo the effects of CHANGE . . . KEEP, so that the configured retention policy applies to the backup.

For example, the following command blocks RMAN from considering backupsets with the tag 'year_end_2002' as obsolete under the retention policy:

```
RMAN> CHANGE BACKUPSET TAG year_end_2002 KEEP;
```

To allow backupsets with the tag year_end_2002 to be marked as obsolete based on the retention policy, use this command:

```
RMAN> CHANGE BACKUPSET TAG year_end_2002 NOKEEP;
```

If you want to prevent the use of a backup marked with KEEP in restore and recovery operations, then mark these backups as UNAVAILABLE. RMAN will not delete the records for these backups from the RMAN repository, but will not try to use them in restore and recovery until they are marked AVAILABLE again.

See Also:

- *Oracle Database Recovery Manager Reference* for CHANGE...KEEP and CHANGE...NOKEEP syntax

Part II

Performing Advanced RMAN Backup and Recovery

Part II describes how to use the RMAN utility to perform advanced backup and recovery operations, including the creation of duplicate and standby databases. It also explains RMAN performance tuning and troubleshooting.

This part contains these chapters:

5

Connecting to Databases with RMAN

This chapter gives detailed instructions for starting the Recovery Manager (RMAN) command-line interface and making database connections. This chapter contains these topics:

Starting RMAN Without Connecting to a Database

You can start RMAN at the operating system command line without connecting to a database by issuing the RMAN command without any arguments. For example, enter:

```
% rman
```

If you did not specify the CMDFILE , SCRIPT or @ option at the command line, then RMAN displays the RMAN prompt:

```
RMAN>
```

After the RMAN prompt is displayed, you can issue further commands to connect to the target database, recovery catalog database, or auxiliary database.

If you start RMAN without specifying either CATALOG or NOCATALOG on the command line, then RMAN makes no recovery catalog connection. The first time a command is issued that requires the RMAN repository, RMAN performs the operation in NOCATALOG mode if you have not connected to a recovery catalog yet. After that point, the CONNECT CATALOG command can no longer be used without exiting and restarting the RMAN client.

Connecting to a Target Database and a Recovery Catalog

The following examples use the following sample connection data, and assume that a recovery catalog is being used.

Value	Description
SYS	User with SYSDBA privileges
oracle	The password for connecting as SYSDBA specified in the target database's password file
trgt	The net service name for the target database
rman	User that owns the recovery catalog schema. This is a user defined in the recovery catalog database that has been granted the RECOVERY_CATALOG_OWNER role.
cat	The password for connecting to the recovery catalog as user RMAN
catdb	The net service name for the recovery catalog database

Connecting to the Target Database and Recovery Catalog from the Command Line

You can specify either operating system or Oracle Net authentication information on the command line when you start the RMAN client.

This example shows how to use operating system authentication to connect to the target database and Oracle Net authentication for the recovery catalog:

```
% rman TARGET / CATALOG rman/cat@catdb
```

This example shows how to use Oracle Net authentication to connect to both the target database and the recovery catalog:

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb
```

Connecting to the Target Database and Recovery Catalog from the RMAN Prompt

You can also start RMAN and connect to the target database from the RMAN prompt. The following example uses operating system authentication for the target database and Oracle Net authentication for the recovery catalog:

```
% rman
RMAN> CONNECT TARGET /
RMAN> CONNECT CATALOG rman/cat@catdb
```

The following example uses Oracle Net password file authentication for the target database, which requires that the target database be using a password file, that defines the password for user SYS to be 'oracle'. Oracle Net authentication is also used for the recovery catalog.

```
% rman
RMAN> CONNECT TARGET SYS/oracle@trgt
RMAN> CONNECT CATALOG rman/cat@catdb
```

> **See Also:** *Oracle Real Application Clusters Administrator's Guide* for details on connecting RMAN to a RAC cluster.

Connecting to an Auxiliary Database

To use the DUPLICATE command, you need to connect to an auxiliary instance. To perform RMAN TSPITR, you may also need to connect to an auxiliary instance if, for example, you do not let RMAN manage the auxiliary instance for you.

See Also:

- Chapter 11, "Duplicating a Database with Recovery Manager" for more details on using the DUPLICATE command
- Chapter 10, "RMAN Tablespace Point-in-Time Recovery (TSPITR)" for more details on performing TSPITR

If the auxiliary instance uses a password file for authentication, then you can connect using a password for either local or remote access. If you are connecting remotely through a net service name, then authentication through a password file is mandatory.

The following dummy values have been substituted into the following examples:

Value	Description
aux	The password for connecting as SYSDBA specified in the auxiliary database's orapwd file
auxdb	The net service name for the auxiliary database

Connecting to an Auxiliary Database from the Command Line

To launch RMAN connected to an auxiliary instance from the operating system command line, enter the following:

```
% rman AUXILIARY SYS/aux@auxdb
```

To connect to target, auxiliary, and recovery catalog databases, launch the RMAN client with these command line arguments:

```
% rman TARGET SYS/oracle@trgt AUXILIARY SYS/aux@auxdb CATALOG rman/cat@catdb
```

Connecting to an Auxiliary Database from the RMAN Prompt

To launch RMAN without connecting to an auxiliary, and connect to the auxiliary database from the RMAN prompt, enter the following commands:

```
% rman
```

```
RMAN> CONNECT AUXILIARY SYS/aux@auxdb
```

To connect to the target, auxiliary, and recovery catalog databases from within RMAN, enter the following commands:

```
% rman
RMAN> CONNECT TARGET SYS/oracle@trgt
RMAN> CONNECT CATALOG rman/cat@catdb
RMAN> CONNECT AUXILIARY SYS/aux@auxdb
```

Diagnosing Connection Problems

When diagnosing errors RMAN encounters in connecting to the target, catalog and auxiliary databases, using SQL*Plus to connect to the databases directly can reveal underlying problems with the connection information or the databases.

Diagnosing Target and Auxiliary Database Connection Problems

RMAN always connects to target and auxiliary databases using the SYSDBA role. Thus, when using SQL*Plus to diagnose connection problems to the target or auxiliary databases, request a SYSDBA connection to reproduce RMAN's behavior.

For example, if the following RMAN command encountered connection errors:

```
RMAN> CONNECT target sys/oracle@target
```

you would reproduce the connection attempt with the SQL*Plus command:

```
SQL> CONNECT sys/oracle@target AS SYSDBA
```

Diagnosing Recovery Catalog Connection Problems

When RMAN connects to the recovery catalog database, it **does not** use the SYSDBA role. So, when you are using SQL*Plus to diagnose connection problems to the recovery catalog database, you must enter the catalog connect string **exactly** as it was entered into RMAN. Do not also specify AS SYSDBA.

Hiding Passwords When Connecting to Databases

If you create an RMAN command file which uses a CONNECT command with database level credentials (user name and password), then anyone with read access

to this file can learn the password. There is no secure way to incorporate a CONNECT string with a password into a command file.

It is also possible, using the ps command under Unix or some similar command under other operating systems, to view command lines and arguments entered into the shell or other host operating system command line interpreter. Therefore, it is risky to invoke RMAN with a command line like this example:

```
% rman TARGET sys/oracle@target
```

To connect to RMAN from the operating system command line and hide authentication information, you can start RMAN without connecting to databases, and then enter CONNECT commands at the RMAN prompt. You can also start RMAN without a password in the connect string, as in this example:

```
% rman TARGET sys@target
```

RMAN will prompt for a password in such a case.

If you create an RMAN command file which uses a CONNECT command that includes authentication information, RMAN does not echo the connect string when you run the command file with the "@" command. This prevents connect strings from appearing in any log files that contain RMAN output.

For example, create a command file listbkup.rman which reads:

```
CONNECT target sys/oracle@target
LIST BACKUP;
```

Then execute this script by running RMAN with the @ command line option:

```
% rman @bkup.rman
```

When the command file executes, RMAN replaces the connection string with an asterisk, as shown in the following output:

```
Recovery Manager: Release 10.1.0.2.0 - Production

Copyright (c) 1995, 2003, Oracle.  All rights reserved.

RMAN> connect target *
2> list backup;
3>
connected to target database: RDBMS (DBID=771530996)

using target database controlfile instead of recovery catalog
```

```
List of Backup Sets
===================
...rest of output omitted
```

Sending RMAN Output Simultaneously to the Terminal and a Log File

If you specify the LOG option at the command line, then RMAN displays command input but does not display the RMAN output. The easiest way to send RMAN output both to a log file and to standard output is to use the UNIX tee command or its equivalent on another operating system. For example:

```
% rman | tee rman.log
RMAN>
```

In this way, both input and output are visible within the RMAN command-line interface.

Executing RMAN Commands Through a Pipe

The RMAN pipe interface is an alternative method for issuing commands to RMAN and receiving the output. By using a pipe, RMAN can interface with the DBMS_PIPE PL/SQL package and avoid the operating system command shell altogether.

If the pipes are not already initialized, then RMAN creates them as private pipes. If you want to put commands on the input pipe before starting RMAN, be careful to first create the pipe by calling DBMS_PIPE.CREATE_PIPE. Whenever a pipe is not explicitly created as a private pipe, the first access to the pipe automatically creates it as a public pipe, and RMAN returns an error if it is told to use a public pipe.

RMAN does not permit the pipe interface to be used with public pipes, because they are a potential security problem. With a public pipe, any user who knows the name of the pipe can send commands to RMAN and intercept its output.

Note: If multiple RMAN sessions can run against the target database, then use unique pipe names for each session of RMAN. The DBMS_PIPE.UNIQUE_SESSION_NAME function is one method that can be used to generate unique pipe names.

Executing Multiple RMAN Commands In Succession Through a Pipe: Example

This scenario assumes that the application controlling RMAN wants to run multiple commands in succession. After each command is sent down the pipe and executed and the output returned, RMAN will pause and wait for the next command.

1. Start RMAN by connecting to a target database (required) and specifying the `PIPE` option. For example, issue:

   ```
   % rman PIPE abc TARGET SYS/oracle@trgt
   ```

   You can also specify the `TIMEOUT` option, which forces RMAN to exit automatically if it does not receive any input from the input pipe in the specified number of seconds. For example, enter:

   ```
   % rman PIPE abc TARGET SYS/oracle@trgt TIMEOUT = 60
   ```

2. Connect to the target database and put the desired commands on the input pipe by using `DBMS_PIPE.PACK_MESSAGE` and `DBMS_PIPE.SEND_MESSAGE`. In pipe mode, RMAN issues message `RMAN-00572` when it is ready to accept input instead of displaying the standard RMAN prompt.
3. Read the RMAN output from the output pipe by using `DBMS_PIPE.RECEIVE_MESSAGE` and `DBMS_PIPE.UNPACK_MESSAGE`.
4. Repeat steps 2 and 3 to execute further commands with the same RMAN instance that was started in step 1.
5. If you used the `TIMEOUT` option when starting RMAN, RMAN terminates automatically after not receiving any input for the specified length of time. To force RMAN to terminate immediately, send the `EXIT` command.

Executing RMAN Commands In a Single Job Through a Pipe: Example

This scenario assumes that the application controlling RMAN wants to run one or more commands as a single job. After running the commands that are on the pipe, RMAN will exit.

1. After connecting to the target database, create a pipe (if it does not already exist under the name `ORA$RMAN_`*`pipe`*`_IN`).
2. Put the desired commands on the input pipe. In pipe mode, RMAN issues message `RMAN-00572` when it is ready to accept input instead of displaying the standard RMAN prompt.
3. Start RMAN with the `PIPE` option, and specify `TIMEOUT = 0`. For example, enter:

```
% rman PIPE abc TARGET SYS/oracle@trgt TIMEOUT = 0
```

4. RMAN reads the commands that were put on the pipe and executes them by using `DBMS_PIPE.PACK_MESSAGE` and `DBMS_PIPE.SEND_MESSAGE`. When it has exhausted the input pipe, RMAN exits immediately.
5. Read RMAN output from the output pipe by using `DBMS_PIPE.RECEIVE_MESSAGE` and `DBMS_PIPE.UNPACK_MESSAGE`.

See Also: *PL/SQL Packages and Types Reference* for documentation on the `DBMS_PIPE` package and "RMAN Pipe Interface" on page 1-6 for a brief overview of RMAN pipes

6

Configuring the RMAN Environment: Advanced Topics

This chapter describes how to perform setup and configuration tasks. This chapter contains these topics:

- Configuring the Flash Recovery Area: Advanced Topics
- Configuring RMAN to Make Backups to a Media Manager
- Configuring Automatic Channels
- Configuring the Maximum Size of Backup Sets and Pieces
- Configuring Backup Optimization
- Configuring Backup Duplexing: CONFIGURE... BACKUP COPIES
- Configuring Tablespaces for Exclusion from Whole Database Backups
- Setting the Snapshot Control File Location
- Setting Up RMAN for Use with a Shared Server

> **See Also:** *Oracle Database Backup and Recovery Basics* for basic RMAN configuration information

Configuring the Flash Recovery Area: Advanced Topics

To take maximum advantage of the flash recovery area, it should be used to store and manage as many different types of file as possible: online redo logs, archived redo logs, control files and RMAN's own working files for backup and restore operations.

This section contains the following topics:

- Configuring Online Redo Log Creation in the Flash Recovery Area
- Configuring Control File Creation in the Flash Recovery Area
- Archived Redo Log Creation in the Flash Recovery Area
- RMAN File Creation in the Flash Recovery Area

Configuring Online Redo Log Creation in the Flash Recovery Area

The following statements can create online redo logs in the flash recovery area:

- `CREATE DATABASE`
- `ALTER DATABASE ADD LOGFILE`
- `ALTER DATABASE ADD STANDBY LOGFILE`
- `ALTER DATABASE OPEN RESETLOGS`

The default size of an online log created in the flash recovery area is 100 MB. The log member filenames are automatically generated by the database.

The initialization parameters that determine where online redo log files are created are `DB_CREATE_ONLINE_LOG_DEST_`*n*, `DB_RECOVERY_FILE_DEST` and `DB_CREATE_FILE_DEST`. Details of the effect of various combinations of these parameters on online redo log creation can be found in*Oracle Database SQL Reference* in the description of the `LOGFILE` clause of the `CREATE DATABASE` statement.

Configuring Control File Creation in the Flash Recovery Area

The initialization parameters `CONTROL_FILES`, `DB_CREATE_ONLINE_LOG_DEST_`*n*, `DB_RECOVERY_FILE_DEST`, and `DB_CREATE_FILE_DEST` all interact to determine the location where the database control files are created.

For a full description of how these parameters interact, see the "Semantics" section of the description of `CREATE CONTROLFILE` in *Oracle Database SQL Reference.*

If the database creates an Oracle managed control file, and if the database uses a server parameter file, then the database sets the CONTROL_FILES initialization parameter in the server parameter file. If the database uses a client-side initialization parameter file, then you must set the CONTROL_FILES initialization parameter manually in the initialization parameter file.

Archived Redo Log Creation in the Flash Recovery Area

It is recommended that you the use flash recovery area as an archived log location because the archived logs are automatically managed by the database. Whatever archiving scheme you choose, it is always advisable to create multiple copies of archived logs.

You have the following basic options, listed from most to least recommended:

1. Enable archiving to the flash recovery area *only* and use disk mirroring to create copies of the archived redo logs.
2. Enable archiving to the flash recovery area and set other LOG_ARCHIVE_DEST_*n* initialization parameter to locations outside the flash recovery area.
3. Set LOG_ARCHIVE_DEST_*n* initialization parameters to archive *only* to non-flash recovery area locations.

If you want to use the flash recovery area, you cannot use the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST initialization parameters. You must use instead the LOG_ARCHIVE_DEST_*n* parameters, which have somewhat different semantics. Once your database is using LOG_ARCHIVE_DEST_*n*, you can configure a flash recovery area.

Rules for Initialization Parameters Affecting Redo Log File Destinations

The interactions among different initialization parameters affecting redo log archiving desintations are as follows:

- If LOG_ARCHIVE_DEST (and, optionally, LOG_ARCHIVE_DUPLEX_DEST) is set, these parameters will specify the only redo log archiving destinations.
- If DB_RECOVERY_FILE_DEST is specified (that is, if a flash recovery area is configured) and no LOG_ARCHIVE_DEST_*n* is specified, then LOG_ARCHIVE_DEST_10 is implicitly set to the flash recovery area. (You can override this behavior by explicitly setting LOG_ARCHIVE_DEST_10 to an empty string.)
- If you set any local destinations for LOG_ARCHIVE_DEST_*n*, then archived redo logs are stored only in the destinations you specify using those parameters. In this case, redo log files are not archived in the flash recovery area by default. If

you have a flash recovery area configured, you can explicitly add the flash recovery area to the set of archiving destinations by setting one of the LOG_ARCHIVE_DEST_*n* parameters to LOCATION=USE_DB_RECOVERY_FILE_DEST (note that this does not have to be LOG_ARCHIVE_DEST_10).

- If you do not set any value for LOG_ARCHIVE_DEST, LOG_ARCHIVE_DEST_*n*, or DB_RECOVERY_FILE_DEST, then the redo logs are archived to a default location that is platform-specific. On Solaris, for example, the default is ?/dbs.

Filenames for Archived Redo Log Files in the Flash Recovery Area

The generated filenames for the archived redo logs in the flash recovery area are Oracle Manged Filenames and are not determined by LOG_ARCHIVE_FORMAT.

RMAN File Creation in the Flash Recovery Area

This section describes RMAN commands or implicit actions (such as control file autobackup) that can create files in the flash recovery area, and how to control whether a specific command creates files there or in some other destination. The assumption in all cases is that a flash recovery area has already been configured for your database. The commands are:

- **BACKUP**

  Do not specify a FORMAT option to the BACKUP command, and do not configure a FORMAT option for disk backups. In such a case, RMAN creates backup pieces and image copies in the flash recovery area, with names in Oracle Managed Files name format.

- **Control File Autobackup**

  RMAN can create control file autobackups in the flash recovery area. Use the RMAN command CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK CLEAR to clear any configured format option for the control file autobackup location on disk. Control file autobackups will be placed in the flash recovery area when no other destination is configured.

- **RESTORE ARCHIVELOG**

  Explicitly or implicitly (as in the case of), set one of the LOG_ARCHIVE_DEST_n) parameters to 'LOCATION=USE_DB_RECOVERY_FILE_DEST'. If you do not specify SET ARCHIVELOG DESTINATION to override this behavior, then restored archived redo log files will be stored in the flash recovery area.

- **RECOVER DATABASE** or **TABLESPACE**, **BLOCKRECOVER**, and **FLASHBACK DATABASE**

These commands restore archived redo logs from backup for use during media recovery, as required by the command. RMAN restores any redo log files needed during these operations to the flash recovery area, and delete them once they are applied during media recovery.

To direct the restored archived redo logs to the flash recovery area, set one of the `LOG_ARCHIVE_DEST_`*n* parameters to '`LOCATION=USE_DB_RECOVERY_FILE_DEST`", and make sure you are not using `SET ARCHIVELOG DESTINATION` to direct restored archived logs to some other destination.

Configuring RMAN to Make Backups to a Media Manager

On most platforms, to back up to and restore from sequential media such as tape you must integrate a media manager with your Oracle database. A media manager is not an Oracle product and must be obtained from a third-party vendor. If you choose to use RMAN with a media manager, then you must obtain all product-specific information from the vendor.

This section describes the generic steps for configuring RMAN for use with a media manager. The actual steps depend on the media management product that you install and the platform on which you are running the database.

Read the following sections in order when configuring the media manager:

1. Prerequisites for Using a Media Manager with RMAN
2. Locating the Media Management Library: The SBT_LIBRARY Parameter
3. Testing Whether the Media Manager Library Is Integrated Correctly
4. Configuring Automatic Channels for Use with a Media Manager

> **See Also:** "Media Management" on page 1-11 for an overview of media management software and its implications for RMAN

Prerequisites for Using a Media Manager with RMAN

Before you can begin using RMAN with a media manager, you must install it and make sure that RMAN can communicate with it. Instructions for this procedure should be available in the media manager vendor's software documentation.

In general, you should begin by installing and configuring the media management software on the target host or production network. Ensure that you can make non-RMAN backups of operating system files on the target database host. This step makes later troubleshooting much easier, by confirming that the basic integration of

the media manager with the target host has been successful. Refer to your media management documentation to learn how to back up files to the media manager outside of RMAN.

Then, obtain and install the third-party media management module for integration with the database server. This module contains the media management library that the Oracle database loads and uses when accessing the media manager. It is generally a third-party product which must be purchased separately. Contact your media management vendor for details.

Locating the Media Management Library: The SBT_LIBRARY Parameter

When allocating or configuring channels for RMAN to use to communicate with a media manager, specify the SBT_LIBRARY parameter to provide the path to the media management software library. When RMAN actually allocates channels to communicate with a media manager, it attempts to load the library indicated by the SBT_LIBRARY parameter.

If you do not provide a value for this parameter, RMAN looks in a platform-specific default location. On UNIX, the default library filename is $ORACLE_HOME/lib/libobk.so, with the extension name varying according to platform: .so, .sl, .a, and so forth. On Windows the default library location is %ORACLE_HOME%\bin\orasbt.dll.

> **Note:** The default media management library file is *not* part of the standard database installation. It is only present if you install third-party media management software.

If the database is unable to locate a media management library in the location specified by the SBT_LIBRARY parameter or the default location, then RMAN issues an ORA-27211 error and exits.

Whenever channel allocation fails, the database writes a trace file to the USER_DUMP_DEST directory. The following shows sample output:

```
SKGFQ OSD: Error in function sbtinit on line 2278
SKGFQ OSD: Look for SBT Trace messages in file /oracle/rdbms/log/sbtio.log
SBT Initialize failed for /oracle/lib/libobk.so
```

See Also:

- Your operating system specific Oracle documentation and the documentation supplied by your media vendor for instructions on how to achieve media manager integration on your platform
- "After Installation of Media Manager, RMAN Channel Allocation Fails: Scenario" on page 15-17 for troubleshooting scenarios involving media manager problems

Testing Whether the Media Manager Library Is Integrated Correctly

After you have confirmed that the database server can load the media management library, test to make sure that RMAN can back up to the media manager. The process for testing the media management library is described in the following sections:

- Configuring Media Management Software for RMAN Backups
- Testing ALLOCATE CHANNEL on the Media Manager
- Testing a Backup to the Media Manager

Configuring Media Management Software for RMAN Backups

After installing the media management software, perform whatever configuration that your vendor requires so that the software can accept RMAN backups. Depending on the type of media management software that you installed, you may have to define media pools, configure users and classes, and so forth.

Then, determine which PARMS settings are needed for the ALLOCATE CHANNEL or CONFIGURE CHANNEL commands as well as the recommended FORMAT string for the BACKUP command (if needed). The PARMS parameter sends instructions to the media manager. For example, the following vendor-specific PARMS setting instructs the media manager to back up to a volume pool called oracle_tapes:

```
PARMS='ENV=(NSR_DATA_VOLUME_POOL=oracle_tapes)'
```

Refer to your third-party vendor documentation for the appropriate settings.

See Also:

- *Oracle Database Recovery Manager Reference* for ALLOCATE CHANNEL syntax
- *Oracle Database Recovery Manager Reference* for channel control options

Configuring Backup Piece Names and Sizes for a Media Manager

To work with restrictions on file names and sizes imposed by your media manager, you may need to configure RMAN settings that control the naming and size of backup pieces.

Configuring Backup Piece Names for RMAN Backups to a Media Manager You may need to manage the naming of backup pieces to be written to the media manager, so that backup pieces have unique names. A backup piece name is determined by the FORMAT string specified in the BACKUP command, the CONFIGURE CHANNEL command, or the ALLOCATE CHANNEL command. The media manager considers the backup piece name as the filename of the backup file, so this name must be unique in the media manager catalog.

You can use the substitution variables provided by RMAN to generate unique backup piece names. If you do not specify the FORMAT parameter, then RMAN automatically generates a unique filename with the %U substitution variable.

Note: Refer to your media management documentation to determine the string character limit for the media manager. For example, some media managers only support a 14-character backup piece name, and some require special FORMAT strings. The unique backup piece names generated by %U are less than 14 characters.

See Also: *Oracle Database Recovery Manager Reference* for the complete list of variables allowable in format strings with the BACKUP command

Configuring Backup Piece Sizes for RMAN Backups to a Media Manager Some media managers have limits on the maximum size of files that they can back up or restore. You must ensure that RMAN does not produce backup sets larger than limits imposed by your media manager.

To limit backup piece sizes, use the parameter MAXPIECESIZE, which you can set in the CONFIGURE CHANNEL and ALLOCATE CHANNEL commands. Refer to the *.rcv scripts in the demo subdirectory on your system, which is located in an

operating system specific location (`$ORACLE_HOME/rdbms` on UNIX) for an example.

> **See Also:** *Oracle Database Recovery Manager Reference* and "Size of Backup Pieces" on page 2-28for details on how to set `MAXPIECESIZE`

Testing ALLOCATE CHANNEL on the Media Manager

Use the following steps to confirm that RMAN is able to load the media management library when allocating a channel for your media manager.

1. Start RMAN and connect to the target database. For example, enter:

```
% rman TARGET /
```

2. Run the `ALLOCATE CHANNEL` command with the `PARMS` required by your media management software. For example, run this command:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt
    PARMS='SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_SERVER=tape_
srv,NSR_GROUP=oracle_tapes)';
}
```

If you do not receive an error message, then the database successfully loaded the media management library. If you receive the `ORA-27211` error, the media management library could not be loaded:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of allocate command on c1 channel at 11/30/2001 13:57:18
ORA-19554: error allocating device, device type: SBT_TAPE, device name:
ORA-27211: Failed to load Media Management Library
Additional information: 25
```

In this case, you must check your media management installation to make sure that the library is correctly installed, and re-check the value for the SBT_LIBRARY parameter as described in "Locating the Media Management Library: The SBT_LIBRARY Parameter" on page 6-6.

For any other errors, check the trace file in `USER_DUMP_DEST` directory for more information.

> **See Also:** "After Installation of Media Manager, RMAN Channel Allocation Fails: Scenario" on page 15-17 for a troubleshooting scenario

Testing a Backup to the Media Manager

After testing a channel allocation on the media manager, make a test backup. For example, to test whether your backup goes successfully to tape, you might run the following command:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt
    PARMS='SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_SERVER=tape_srv,NSR_
GROUP=oracle_tapes)';
  BACKUP CURRENT CONTROLFILE;
}
```

The specifics of your PARMS and FORMAT settings depend on the media management software that you are using.

If the backup succeeds, then you are ready to make backups to your media manager.

Possible failures include the following cases:

Case	Response
The backup hangs.	A hanging backup usually indicates that the media manager is waiting to mount a tape. Check if there are any media manager jobs in "tape mount request" mode and fix the problem. Ensure that the steps in "Configuring RMAN to Make Backups to a Media Manager" on page 6-5 are correctly done. Refer to "Backup Job Is Hanging: Scenario" on page 15-19 if the problem persists.
The backup fails with an ORA-19511	This error indicates that the media management software is not correctly configured. Ensure that the steps in "Configuring RMAN to Make Backups to a Media Manager" on page 6-5 are correctly done. Also, ensure that you have the correct PARMS and FORMAT strings required by your media management software.

> **See Also:** "Testing the Media Management API" on page 15-10 and "RMAN Troubleshooting Scenarios" on page 15-16 for more information about troubleshooting RMAN with a media manager

Configuring Automatic Channels for Use with a Media Manager

This section describes how to configure automatic channels specifically for use with a media manager. For an overview of automatic channels and how they are used, refer to the section "Configuring Automatic Channels" on page 6-12. The following setup procedure references the sections in "Configuring Automatic Channels" where it is appropriate.

To configure automatic channels for use with a media manager:

1. Configure a generic channel of DEVICE TYPE sbt as described in "Configuring a Generic Automatic Channel for a Device Type" on page 6-13. In the configuration enter all parameters that you tested in the section "Testing a Backup to the Media Manager" on page 6-10. For example, assume that your media vendor requires PARMS settings as follows:

   ```
   RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt
   PARMS='SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_SERVER=tape_svr,NSR_
   CLIENT=oracleclnt,NSR_GROUP=ora_tapes)'
     FORMAT "BACKUP_%U";
   ```

2. After configuring the channel, test by backing up something small, such as the control file:

   ```
   RMAN> BACKUP DEVICE TYPE sbt CURRENT CONTROLFILE;
   ```

3. Check your configuration by running the following command:

   ```
   RMAN> SHOW CHANNEL FOR DEVICE TYPE sbt;
   ```

4. Configure the default device to sbt so that RMAN sends all backups to the media manager. For example:

   ```
   RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
   ```

5. After configuring the default device, make a test backup to determine whether it is really going to the media manager:

   ```
   RMAN> BACKUP CURRENT CONTROLFILE;
   ```

6. Check your configuration by running the following command:

   ```
   RMAN> SHOW DEFAULT DEVICE TYPE;
   ```

7. If you use more than one tape device, then you must specify the channel parallelism as described in "Configuring Parallelism for Automatic Channels"

on page 6-12. Assume that you want to back up to your media manager using two tape drives in parallel. In this case, you can run the following commands:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 2;
RMAN> BACKUP DATABASE;
```

Configuring Automatic Channels

You can save persistent configuration information such as channel parameters, parallelism, and the default device type in the RMAN repository. Hence, you do not have to manually allocate channels for each backup. Instead, you can configure automatic channels for use in backup, restore, recovery, and maintenance jobs.

You can always override automatic channels with ALLOCATE CHANNEL to allocate channels manually for a particular backup job.

By default, RMAN has preconfigured a disk channel so that you can back up to disk without doing any manual configuration. You may, however, want to parallelize the channels for disk or tape devices to improve performance.

> **See Also:** "About RMAN Channels" on page 2-2 for a conceptual overview of automatic and manual channels, and *Oracle Database Recovery Manager Reference* for syntax

Configuring Parallelism for Automatic Channels

By default, channel parallelism for each configured device is set to 1. As a rule, allocate one channel for each physical device. If you are backing up to only one disk location or only one tape drive, then you need only one channel.

The CONFIGURE DEVICE TYPE . . . PARALLELISM *integer* command specifies how many channels (up to 254) RMAN should allocate for jobs on the specified device type. This command allocates three channels for jobs on device type DISK :

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
```

These commands back up to a media manager using two tape drives in parallel:

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt; # default backup device is tape
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 2; # configure two tape channels
RMAN> BACKUP DATABASE; # backup goes to two tapes, in two parallel streams
```

Each configured sbt channel will back up roughly half the total data.

See Also: "Determining Channel Parallelism to Match Hardware Devices" on page 2-9

Configuring a Generic Automatic Channel for a Device Type

By default, RMAN automatically allocates a preconfigured DISK channel without any options. However, you may use a media manager that requires special options (PARMS, FORMAT, MAXPIECESIZE, and so forth) or you may want to change the default DISK setting. By configuring channels, you define which parameters are used when RMAN automatically allocates channels.

Use the CONFIGURE CHANNEL command to configure automatic channel options for the available device types: DISK and sbt. You can use the same options for CONFIGURE CHANNEL that you use for ALLOCATE CHANNEL, and you must specify at least one of these options. For example, you can configure generic disk and tape channels as in this example:

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT = '?/bkup_%U';
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt
  PARMS='SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_SERVER=tape_svr,NSR_
CLIENT=oracleclnt,NSR_GROUP=ora_tapes)';
```

To configure a **generic channel**, that is, a template that is used for all parallelized channels, do not assign a number for the channel. If you set the PARALLELISM for a device, and then make the device default, then RMAN uses the same channel configuration for each parallelized channel.

To configure new generic channel settings for a specified device type, simply run a new command for the device type. The following example configures the default DISK channel to MAXPIECESIZE 2G, then erases this setting and sets a FORMAT:

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 2G;
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT = /tmp/%U;
```

Configured Channels and the Default Device Type

The automatic channel that RMAN allocates for its backups depends on the default device type. If the default device type is DISK, then RMAN uses the DISK channel only. If the default device type is sbt, then RMAN uses the sbt channel only. RMAN cannot automatically allocate channels in backup jobs for multiple device types simultaneously (and, in fact, you should never attempt to use channels for multiple device types simultaneously for any backup job).

The following example creates a configuration in which all backups go to two tapes in parallel. For this example, the media management software requires additional

parameters besides specifying the SBT_LIBRARY: ENV=(NSR_DATA_VOLUME_POOL=oracle_tapes). The chosen FORMAT for backup file names is %U_backup.

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;        # by default, backup goes to MML
CONFIGURE DEVICE TYPE sbt PARALLELISM 2;     # two tapes in parallel
CONFIGURE CHANNEL DEVICE TYPE                # sets parameters for all channels
  PARMS 'SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_DATA_VOLUME_
POOL=oracle_tapes)' FORMAT '%U_backup';
BACKUP DATABASE;                             # backs up database
```

Showing the Automatic Channel Configuration Settings

The SHOW CHANNEL, SHOW DEVICE TYPE and SHOW DEFAULT DEVICE TYPE commands are used to display the current configured settings for automatic channels.

Showing the Automatic Channel Settings

After connecting to the target database and recovery catalog (if you use one), issue the SHOW CHANNEL command to display the settings for all automatically allocated channels. For example, connect the RMAN client to the target and possibly catlaog databases, and enter:

```
RMAN> SHOW CHANNEL;    # shows the CONFIGURE setting for the automatic channels
```

Sample output for SHOW CHANNEL follows:

```
RMAN configuration parameters are:
CONFIGURE CHANNEL DEVICE TYPE 'SBT' RATE 1500K;
```

Showing the Configured Device Types

Issue the SHOW DEVICE TYPE command to display the configured devices and their PARALLELISM and backup type settings. The DISK device type is preconfigured.

To show the default device type and configuration for automatic channels:

After connecting to the target database and recovery catalog (if you use one), run the SHOW DEVICE TYPE command. For example, enter:

```
SHOW DEVICE TYPE;    # shows the CONFIGURE DEVICE TYPE ... PARALLELISM settings
```

Sample output for SHOW DEVICE TYPE follows:

```
RMAN configuration parameters are:
```

```
CONFIGURE DEVICE TYPE 'SBT_TAPE' PARALLELISM 1 BACKUP TYPE TO BACKUPSET;
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO COMPRESSED BACKUPSET; # default
```

> **Note:** As with all SHOW commands, the output of SHOW DEVICE TYPE is in the form of a valid RMAN CONFIGURE command. You can in fact enter one command, like those shown in the preceding sample output, to configure the backup type and parallelism simultaneously. Refer to the syntax diagrams for CONFIGURE in *Oracle Database Recovery Manager Reference* for details on all of the possible ways of combining arguments to the CONFIGURE command.

Showing the Default Device Type

Issue the SHOW DEFAULT DEVICE TYPE command to display the settings for the default device type used by the automatic channels. When you issue the BACKUP command, RMAN allocates only default channels of the type set by the CONFIGURE DEFAULT DEVICE TYPE command. This default device type setting is not in effect when you use commands other than BACKUP. Note that you cannot disable the default device type: it is always either DISK (default setting) or sbt.

To show the default device type for automatic channels:

After connecting to the target database and recovery catalog (if you use one), run the SHOW DEFAULT DEVICE TYPE command. For example, enter:

```
SHOW DEFAULT DEVICE TYPE;    # shows the CONFIGURE DEFAULT DEVICE TYPE setting
```

Sample output for SHOW DEFAULT DEVICE TYPE follows:

```
RMAN configuration parameters are:
CONFIGURE DEFAULT DEVICE TYPE TO 'SBT';
```

Manually Overriding Configured Channels

If you manually allocate a channel during a job, then RMAN disregards any automatic channel settings. For example, assume that the default device type is configured to sbt, and you execute this command:

```
RMAN> RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE DISK;
  BACKUP TABLESPACE users;
}
```

In this case, RMAN uses only the disk channel that you manually allocated within the RUN block, overriding any defaults set by using CONFIGURE DEVICE TYPE, CONFIGURE DEFAULT DEVICE, or CONFIGURE CHANNEL settings.

See Also:

- "About RMAN Channels" on page 2-2 to learn about automatic channels
- *Oracle Database Recovery Manager Reference* for ALLOCATE syntax
- *Oracle Database Recovery Manager Reference* for CONFIGURE syntax

Configuring a Specific Channel for a Device Type

Besides configuring a generic channel for a device, you can also configure one or more specific channels for each device type by manually assigning your own channel numbers to the channels. Run the CONFIGURE CHANNEL *n* command (where *n* is a positive integer less than 255) to configure a specific channel. When manually numbering channels, you must specify one or more channel options (for example, MAXPIECESIZE or FORMAT) for each channel. When you use that specific numbered channel in a backup, the configured settings for that channel will be used instead of the configured generic channel settings.

Configure specific channels by number when it is necessary to control the parameters set for each channel separately. This could arise in the following situations:

- When running a Real Application Clusters (RAC) configuration, in which multiple nodes back up the cluster requiring different connect strings
- When running a Real Application Cluster and using a media manager with multiple tape drives requiring different PARMS settings

Configuring Specific Channels: Examples

For example, assume that you have two tape drives and want one tape drive to use tapes from the first pool and the second tape drive to use tapes from second tape pool. You run the following commands:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;    # backup goes to sbt
CONFIGURE DEVICE TYPE sbt PARALLELISM 2; # two tapes used in parallel
# configure first stream to go to data volume pool named first_pool
```

```
CONFIGURE CHANNEL 1 DEVICE TYPE sbt
  PARMS 'SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_DATA_VOLUME_POOL=first_
pool)';
# configure second stream to go to data volume pool named second_pool
CONFIGURE CHANNEL 2 DEVICE TYPE sbt
  PARMS 'SBT_LIBRARY=/mediavendor/lib/libobk.so ENV=(NSR_DATA_VOLUME_
POOL=second_pool)';
BACKUP DATABASE; # first stream goes to 'first_pool' and second to 'second_pool'
```

In this example, you want to back up to two different disks because not enough space exists on a single disk. So, you do the following:

```
CONFIGURE DEFAULT DEVICE TYPE TO disk;        # backup goes to disk
CONFIGURE DEVICE TYPE sbt PARALLELISM 2;      # two channels used in in parallel
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '/disk1/%U' # 1st channel to disk1
CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT '/disk2/%U' # 2nd channel to disk2
BACKUP DATABASE; # backup - first channel goes to disk1 and second to disk2
```

Mixing Generic and Specific Channels

When parallelizing, RMAN always allocates channels beginning with CHANNEL 1 and ending with channel number equal to the PARALLELISM setting. Hence, RMAN uses a specific configuration for a given channel if you have configured it; otherwise, it uses a generic configuration.

Assume you enter the following channel configuration:

```
# disk channel configuration
CONFIGURE DEVICE TYPE DISK PARALLELISM 4;
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT = '/tmp/backup_%U';
CONFIGURE CHANNEL 2 DEVICE TYPE DISK MAXPIECESIZE = 20M;
CONFIGURE CHANNEL 4 DEVICE TYPE DISK MAXPIECESIZE = 40M;

# sbt channel configuration
CONFIGURE DEVICE TYPE sbt PARALLELISM 3;
CONFIGURE CHANNEL DEVICE TYPE sbt
      PARMS='SBT_LIBRARY=oracle.disksbt, ENV=(BACKUP_DIR=?/oradata)';
CONFIGURE CHANNEL 3 DEVICE TYPE sbt
      PARMS='SBT_LIBRARY=oracle.disksbt, ENV=(BACKUP_DIR=/tmp)';
```

The following table illustrates the channel names and channel settings that RMAN allocates when the default device is DISK and PARALLELISM for DISK is set to 4.

Channel Name	Setting
ORA_DISK_1	FORMAT = '/tmp/backup_%U'
ORA_DISK_2	MAXPIECESIZE = 20M
ORA_DISK_3	FORMAT = '/tmp/backup_%U'
ORA_DISK_4	MAXPIECESIZE = 40M

The following table illustrates the channel names and channel settings that RMAN allocates when the default device is sbt and PARALLELISM for sbt is set to 3.

Channel Name	Setting
ORA_SBT_TAPE_1	PARMS='ENV=(BACKUP_DIR=?/oradata)'
ORA_SBT_TAPE_2	PARMS='ENV=(BACKUP_DIR=?/oradata)'
ORA_SBT_TAPE_3	PARMS='ENV=(BACKUP_DIR=/tmp)'

Relationship Between CONFIGURE CHANNEL and Parallelism Setting

The PARALLELISM setting is not constrained by the number of specifically configured channels. For example, if you back up to 20 different tape devices, then you can configure 20 different sbt channels, each with a manually assigned number (from 1 to 20) and each with a different set of channel options. In such a situation, you can set PARALLELISM to any value up to the number of devices, in this instance 20.

RMAN always numbers parallel channels starting with 1 and ending with the PARALLELISM setting. For example, if the default device is sbt and PARALLELISM for sbt is set to 3, then RMAN names the channels as follows:

```
ORA_SBT_TAPE_1
ORA_SBT_TAPE_2
ORA_SBT_TAPE_3
```

RMAN always uses the name ORA_SBT_TAPE_*n* even if you configure DEVICE TYPE sbt (not the synonymous sbt_tape). RMAN always allocates the number of channels specified in PARALLELISM, using specifically configured channels if you have configured them and generic channels if you have not.

See Also: "Automatic Channel-Specific Configurations" on page 2-8 for concepts about manually numbered channels, and "Configuring Specific Channels: Examples" on page 6-16

Clearing Channel and Device Settings

To clear a configuration is to return it to its default settings. You can clear channel and device settings by using these commands:

- `CONFIGURE DEVICE TYPE ... CLEAR`
- `CONFIGURE DEFAULT DEVICE TYPE CLEAR`
- `CONFIGURE CHANNEL DEVICE TYPE ... CLEAR`
- `CONFIGURE CHANNEL` *n* `DEVICE TYPE ... CLEAR` (where *n* is an integer)

Each `CONFIGURE ... CLEAR` command clears only itself. For example, `CONFIGURE DEVICE TYPE ... CLEAR` does not clear `CONFIGURE DEFAULT DEVICE TYPE`. The `CONFIGURE DEVICE TYPE ... CLEAR` command removes the configuration for the specified device type and returns it to the default (`PARALLELISM 1`).

Note: You cannot specify any other options when clearing a device type.

The `CONFIGURE DEFAULT DEVICE TYPE ... CLEAR` command clears the configured default device and returns it to `DISK` (the default setting).

The `CONFIGURE CHANNEL DEVICE TYPE ... CLEAR` command erases the channel configuration for the specified device type. RMAN does not change the `PARALLELISM` setting for the device type because `PARALLELISM` is specified through a separate `CONFIGURE` command.

If you have manually assigned options to automatic channels, then clear the options for these channels individually by specifying the channel number in `CONFIGURE CHANNEL` *n* `DEVICE TYPE ... CLEAR`. For example, assume that you run the following:

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE = 1800K;
RMAN> CONFIGURE CHANNEL 3 DEVICE TYPE DISK FORMAT = /tmp/%U;
RMAN> CONFIGURE CHANNEL 3 DEVICE TYPE DISK CLEAR;
```

In this case, RMAN clears the settings for `CHANNEL 3`, but leaves the settings for the generic `DISK` channel (the channel with no number manually assigned) intact.

See Also: "Clearing Automatic Channel Settings" on page 2-8

Configuring the Maximum Size of Backup Sets and Pieces

The CONFIGURE MAXSETSIZE command limits the size of backup sets created on a channel. This CONFIGURE setting applies to any channel, whether manually or automatically allocated, when the BACKUP command is used to create backup sets.

You can set MAXSETSIZE in bytes (default), kilobytes (K), megabytes (M), and gigabytes (G). The default value is given in bytes and is rounded down to the lowest kilobyte value. For example, if you set the maximum set size to 2000, then RMAN rounds down this value to 1 kilobyte (1024 bytes). If you set the maximum set size to 2049, then RMAN rounds down this value to 2 kilobytes (2048 bytes).

The value set by the CONFIGURE MAXSETSIZE command is a default for the given channel. You can override the configured MAXSETSIZE value by specifying a MAXSETSIZE option for an individual BACKUP command.

Assume that you issue the following commands at the RMAN prompt:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
CONFIGURE CHANNEL DEVICE TYPE sbt PARMS 'ENV=(NSR_DATA_VOLUME_POOL=first_pool)';
CONFIGURE MAXSETSIZE TO 7500K;
BACKUP TABLESPACE users;
BACKUP TABLESPACE tools MAXSETSIZE 5G;
```

The results will be as follows:

- The backup of the users tablespace uses the automatic sbt channel and the configured default MAXSETSIZE setting of 7500K.
- The backup of the tools tablespace uses the MAXSETSIZE setting of 5G used in the BACKUP command.

Note: There is no equivalent to MAXSETSIZE for controlling the size of image copies. Since an image copy is an exact duplicate of the file being backed up, its size must be identical to the source file.

This fact can present a problem with some older operating systems which limit the size of individual files. If you are using a raw partition to store a 10GB datafile, and your operating system only supports 4GB files on the file system, you cannot take image copy backups of that file.

See Also: *Oracle Database Recovery Manager Reference* for BACKUP syntax

Showing the Default Maximum Size of Backup Sets: SHOW MAXSETSIZE

You can use SHOW MAXSETSIZE to view the maximum backup set size set using CONFIGURE MAXSETSIZE. The size of a backup set is measured in the total bytes of the included backup pieces. After connecting to the target database and recovery catalog (if you use one), issue the SHOW MAXSETSIZE command. For example, enter:

```
SHOW MAXSETSIZE;      # shows the CONFIGURE MAXSETSIZE settings
```

Sample output for SHOW MAXSETSIZE follows:

```
RMAN configuration parameters are:
CONFIGURE MAXSETSIZE TO 3072K;
```

Configuring Backup Optimization

Run the CONFIGURE command to enable and disable backup optimization. Backup optimization skips the backup of files in certain circumstances if the identical file or an identical version of the file has already been backed up. Full details on the backup optimization algorithm are provided in "Backup Optimization" on page 2-16.

Note that backup optimization applies only to the following commands:

- BACKUP DATABASE
- BACKUP ARCHIVELOG with ALL or LIKE options
- BACKUP BACKUPSET ALL

You can override optimization at any time by specifying the FORCE option on the BACKUP command. For example, you can run:

```
BACKUP DATABASE FORCE;
BACKUP ARCHIVELOG ALL FORCE;
```

By default, backup optimization is configured to OFF. To enable backup optimization, run the following command:

```
CONFIGURE BACKUP OPTIMIZATION ON;
```

To disable backup optimization, run the following command:

```
CONFIGURE BACKUP OPTIMIZATION OFF;
```

To clear the current backup optimization setting, that is, return backup optimization to its default setting of OFF, run this command:

```
CONFIGURE BACKUP OPTIMIZATION CLEAR;
```

See Also:

- "Backup Optimization Algorithm" on page 2-49 for the complete criteria that determine whether a file is identical and the conditions under which backup optimization is operative
- "Backing Up Files Using Backup Optimization" on page 7-9 for examples of how to optimize RMAN backups

Displaying Backup Optimization Setting: SHOW BACKUP OPTIMIZATION

You can use SHOW BACKUP OPTIMIZATION to view the current settings of backup optimization as configured with the CONFIGURE BACKUP OPTIMIZATION command. After connecting to the target database and recovery catalog (if you use one), issue the SHOW BACKUP OPTIMIZATION command. For example, enter:

```
SHOW BACKUP OPTIMIZATION;
```

Sample output for SHOW BACKUP OPTIMIZATION follows:

```
RMAN configuration parameters are:
CONFIGURE BACKUP OPTIMIZATION ON;
```

Configuring Backup Duplexing: CONFIGURE... BACKUP COPIES

Use the CONFIGURE . . . BACKUP COPIES command to specify how many copies of each backup piece should be created on the specified device type for the specified type of file. This feature is known as **duplexing**. The CONFIGURE settings applies only to backup sets of datafiles (which includes the current control file) and archived redo logs. It does not affect image copies.

Note: Control file autobackups on disk are a special case and are *never* duplexed: RMAN always creates one and only one copy.

To configure the number of backup set copies, specify an integer. The following examples show possible configurations:

```
# Makes 2 disk copies of each datafile and control file backup set
# (autobackups excluded)
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 2;
# Makes 3 copies of every archived redo log backup to tape
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE sbt TO 3;
```

If you use the duplexing feature in conjunction with multiple FORMAT strings, then you can name each individual backup set copy. For example, assume that you configure BACKUP COPIES to 3. Then, you can issue:

```
BACKUP DATABASE FORMAT '/tmp/%U', '?/dbs/%U', '?/oradata/%U';
```

RMAN generates 3 identical copies of each backup piece in the backup set, and names each piece according to the specified FORMAT string: the first copy is placed in the /tmp directory, the second in the ?/dbs directory, and the third in the ?/oradata directory. Note that you can specify the FORMAT string on the BACKUP, CONFIGURE CHANNEL, and ALLOCATE CHANNEL commands.

To return a BACKUP COPIES configuration to its default value, run the same CONFIGURE command with the CLEAR option, as in this example:

```
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE sbt CLEAR;
```

By default, CONFIGURE . . . BACKUP COPIES is set to 1 for each device type.

Note: If you do not want to create a persistent copies configuration, then you can specify copies with the BACKUP COPIES and SET BACKUP COPIES commands.

See Also:

- "Manual Parallelization of Backups" on page 2-18 for concepts
- *Oracle Database Recovery Manager Reference* for BACKUP syntax
- *Oracle Database Recovery Manager Reference* for CONFIGURE syntax
- *Oracle Database Recovery Manager Reference* for SET syntax

Showing the Configured Degree of Duplexing: SHOW... BACKUP COPIES

SHOW... BACKUP COPIES lets you view how you have used CONFIGURE ... BACKUP COPIES command to set the number of identical copies that RMAN makes of each of several types of backup.

After connecting to the target database and recovery catalog (if you use one), run the SHOW ARCHIVELOG BACKUP COPIES or SHOW DATAFILE BACKUP COPIES commands. For example, enter:

```
SHOW DATAFILE BACKUP COPIES;    # shows CONFIGURE DATAFILE BACKUP COPIES setting
```

Sample output for SHOW DATAFILE BACKUP COPIES follows:

```
RMAN configuration parameters are:
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE SBT_TAPE TO 1; # default
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
```

Configuring Tablespaces for Exclusion from Whole Database Backups

You can run CONFIGURE EXCLUDE FOR TABLESPACE to exempt the specified tablespace from the BACKUP DATABASE command. The exclusion condition applies to any datafiles that you add to this tablespace in the future.

This tablespace exclusion feature is useful when you do not want to make a specified tablespace part of the regular backup schedule, as in these cases:

- A tablespace is easy to rebuild, so it is more cost-effective to rebuild it than back it up every day.
- A tablespace contains temporary or test data that you do not need to back up.
- A tablespace does not change often and therefore should be backed up on a different schedule from other backups.

For example, you can exclude testing tablespaces cwmlite and example from whole database backups as follows:

```
CONFIGURE EXCLUDE FOR TABLESPACE cwmlite;
CONFIGURE EXCLUDE FOR TABLESPACE example;
```

If you run the following command, then RMAN backs up all tablespaces in the database except cwmlite and example:

```
BACKUP DATABASE;
```

You can still back up the configured tablespaces by explicitly specifying them in a BACKUP command or by specifying the NOEXCLUDE option on a BACKUP DATABASE command. For example, you can enter one of the following commands:

```
# backs up the whole database, including cwmlite and example
BACKUP DATABASE NOEXCLUDE;
BACKUP TABLESPACE cwmlite, example;  # backs up only cwmlite and example
```

You can disable the exclusion feature for `cwmlite` and `example` as follows:

```
CONFIGURE EXCLUDE FOR TABLESPACE cwmlite CLEAR;
CONFIGURE EXCLUDE FOR TABLESPACE example CLEAR;
```

RMAN includes these tablespaces in future whole database backups.

Showing the Tablespaces Excluded from Backups

SHOW EXCLUDE shows how you have used the CONFIGURE EXCLUDE command to exclude tablespaces from whole database backups.

After connecting to the target database and recovery catalog (if you use one), run the SHOW EXCLUDE command. For example, enter:

```
RMAN> SHOW EXCLUDE;    # shows the CONFIGURE EXCLUDE setting
```

Sample output for SHOW EXCLUDE follows:

```
RMAN configuration parameters are:
CONFIGURE EXCLUDE FOR TABLESPACE 'OLD_ACCOUNTS';
```

See Also:

- *Oracle Database Recovery Manager Reference* for BACKUP syntax
- *Oracle Database Recovery Manager Reference* for CONFIGURE syntax

Configuring Auxiliary Instance Datafile Names: CONFIGURE AUXNAME

When performing tablespace point-in-time recovery (TSPITR) or duplicating a database using RMAN, you may want to set the names of datafiles in the auxiliary instance before startingthe actual TSPITR or database duplication.

The command for doing so is:

```
CONFIGURE AUXNAME FOR datafileSpec TO 'filename';
```

where *datafileSpec* identifies some datafile by its original name or datafile number, and *filename* is the new path for the specified file.

For example, you might configure a new auxiliary name for datafile 2 as follows:

```
CONFIGURE AUXNAME FOR DATAFILE 2 TO '/newdisk/datafiles/df2.df;'
```

As with other settings, this CONFIGURE setting is persistent across RMAN sessions until cleared using CONFIGURE ... CLEAR, as shown here:

```
CONFIGURE AUXNAME FOR DATAFILE 2 CLEAR;
```

If you are performing TSPITR or running the DUPLICATE command, then by using CONFIGURE AUXNAME you can preconfigure the filenames for use on the auxiliary database without manually specifying the auxiliary filenames during the procedure.

When renaming files with the DUPLICATE command, CONFIGURE AUXNAME is an alternative to SET NEWNAME. The difference is that after you set the AUXNAME the first time, you do not need to reset the filename when you issue another DUPLICATE command: the AUXNAME setting remains in effect until you issue CONFIGURE AUXNAME ... CLEAR. In contrast, you must reissue the SET NEWNAME command every time you rename files.

See Chapter 10, "RMAN Tablespace Point-in-Time Recovery (TSPITR)" for more details on using CONFIGURE AUXNAME in connection with TSPITR, and Chapter 11, "Duplicating a Database with Recovery Manager" for more on using CONFIGURE AUXNAME in performing database duplication.

Showing the Default Filenames Configured for Auxiliary Channels

To view auxiliary datafile names currently configured for your database, you can use the SHOW AUXNAME command:

```
RMAN> SHOW AUXNAME;
```

Sample output follows:

```
RMAN configuration parameters are:
CONFIGURE AUXNAME FOR DATAFILE '/oracle/oradata/trgt/tools01.dbf' TO
                              '/tmp/tools01.dbf';
```

Setting the Snapshot Control File Location

When RMAN needs to resynchronize from a read-consistent version of the control file, it creates a temporary **snapshot control file**. RMAN needs a snapshot control

file only when resynchronizing with the recovery catalog or when making a backup of the current control file.

The default value for the snapshot control file is platform-specific and depends on the Oracle home. For example, the default filename on some UNIX platforms in Oracle Database 10*g* is `$ORACLE_HOME/dbs/snapcf_@.f`. Note that if you have a flash recovery area configured, the default location for the snapshot control file is *not* the flash recovery area.

In general, you should only need to set the control file location when you are upgrading to the current release from a release earlier than 8.1.7. In these earlier releases, the default location for the snapshot control file was not dependent on the Oracle home, whereas in the current release the default location is dependent on the Oracle home.

Default Location of the Snapshot Control File

By default, the location of the snapshot control file is determined by the rules in the following table:

If you ...	Then ...
Create a new database in the current release	The snapshot control file location uses the default value. In this case, the default snapshot control file location changes if you change the Oracle home.
Upgrade to the current release from a release prior to 8.1.7	The snapshot control file location is not set to the default value. Instead, RMAN uses the snapshot location that is already stored in the control file. In this case, the snapshot control file location does not change if you change the Oracle home.

Viewing the Configured Location of the Snapshot Control File

You can see the current snapshot location by running the `SHOW` command. This example shows a snapshot location that is determined by the default rule:

```
RMAN> SHOW SNAPSHOT CONTROLFILE NAME;
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/oracle/dbs/snapcf_trgt.f'; # default
```

This example shows a snapshot control file that has a nondefault filename:

```
RMAN>  SHOW SNAPSHOT CONTROLFILE NAME;
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/oracle/oradata/trgt/snap_trgt.ctl';
```

Setting the Location of the Snapshot Control File

Use the CONFIGURE SNAPSHOT CONTROLFILE NAME TO '*filename*' command to change the name of the snapshot control file. Subsequent snapshot control files that RMAN creates use the specified filename.

For example, start RMAN and then enter:

```
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/oracle/oradata/trgt/snap_trgt.ctl';
```

You can also set the snapshot control file name to a raw device:

```
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/dev/vgd_1_0/rlvt5';
```

If one RMAN job is already backing up the control file while another needs to create a new snapshot control file, you may see the following message:

```
waiting for snapshot controlfile enqueue
```

Under normal circumstances, a job that must wait for the control file enqueue waits for a brief interval and then successfully retrieves the enqueue. Recovery Manager makes up to five attempts to get the enqueue and then fails the job. The conflict is usually caused when two jobs are both backing up the control file, and the job that first starts backing up the control file waits for service from the media manager.

> **See Also:** "Backup Fails Because of Control File Enqueue: Scenario" on page 15-25, *Oracle Real Application Clusters Administrator's Guide* for handling of snapshot control files in RAC configurations, and *Oracle Database Recovery Manager Reference* for CONFIGURE syntax

To reset the snapshot control file location to the default, run the CONFIGURE SNAPSHOT CONTROLFILE LOCATION CLEAR command.

Showing the Current Snapshot Control File Name

Issue the SHOW SNAPSHOT CONTROLFILE command to display the value set by CONFIGURE SNAPSHOT CONTROLFILE NAME.

Note: In releases prior to Oracle9*i*, the CONFIGURE SNAPSHOT CONTROLFILE command was called SET SNAPSHOT CONTROLFILE.

To show the snapshot control file filename:

After connecting to the target database and recovery catalog (if you use one), run the SHOW SNAPSHOT CONTROLFILE command. For example, enter:

```
SHOW SNAPSHOT CONTROLFILE NAME;   # shows CONFIGURE SNAPSHOT CONTROLFILE setting
```

Sample output for SHOW SNAPSHOT CONTROLFILE follows:

```
RMAN configuration parameters are:
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/oracle/dbs/cf_snap.f';
```

> **See Also:** "Setting the Snapshot Control File Location" on page 6-26 to learn about the snapshot control file and its function

Setting Up RMAN for Use with a Shared Server

RMAN cannot connect to the target database through a shared server dispatcher. RMAN requires a dedicated server process. Nevertheless, you can connect specified sessions to dedicated servers, even when the target is configured for a shared server.

To ensure that RMAN does not connect to a dispatcher when the target database is configured for a shared server, the net service name used by RMAN must include (SERVER=DEDICATED) in the CONNECT_DATA attribute of the connect string.

Oracle Net configuration varies greatly from system to system. The following procedure illustrates only one method. This scenario assumes that the following service name in the tnsnames.ora connects to the target database using the shared server architecture, where inst1 is a value of the SERVICE_NAMES initialization parameter:

```
inst1_shs =
  (DESCRIPTION=
    (ADDRESS=(PROTOCOL=tcp)(HOST=inst1_host)(port1521))
    (CONNECT_DATA=(SERVICE_NAME=inst1)(SERVER=shared))
  )
```

To use RMAN with a shared server:

1. Create a net service name in the tnsnames.ora file that connects to the nonshared SID. For example, enter:

   ```
   inst1_ded =
     (DESCRIPTION=
       (ADDRESS=(PROTOCOL=tcp)(HOST=inst1_host)(port1521))
       (CONNECT_DATA=(SERVICE_NAME=inst1)(SERVER=dedicated))
     )
   ```

2. Start SQL*Plus and then connect using both the shared server and dedicated server service names to confirm the mode of each session. For example, to connect to a dedicated session you can issue:

```
CONNECT SYS/oracle@inst1_ded
SELECT SERVER
  FROM V$SESSION
  WHERE SID = (SELECT DISTINCT SID FROM V$MYSTAT);

SERVER
---------
DEDICATED
1 row selected.
```

   To connect to a shared server session, you can issue:

```
CONNECT SYS/oracle@inst1_shs AS SYSDBA
SELECT SERVER
  FROM V$SESSION
  WHERE SID = (SELECT DISTINCT SID FROM V$MYSTAT);

SERVER
---------
SHARED
1 row selected.
```

3. Connect to the target database (and optionally the recovery catalog) with the dedicated service name. For example, enter:

```
% rman TARGET SYS/oracle@inst1_ded CATALOG rman/cat@catdb
```

   See Also: Your operating system-specific Oracle documentation and your *Oracle Net Services Reference Guide* for a complete description of Oracle Net connect string syntax

7

Making Backups with RMAN: Advanced Topics

This chapter describes how to use RMAN to make backups. This chapter contains these topics:

Configuring and Allocating Channels for Use in Backups

You have the following options for executing backups:

- Configure automatic channels with the CONFIGURE command, and then issue BACKUP commands at the RMAN prompt or within a RUN block
- Within a RUN block only, you can allocate channels manually with the ALLOCATE CHANNEL command, and then issue BACKUP commands using those channels

The easiest way to make backups is to configure automatic channels. For example, so long as you have already configured an sbt device type, you can configure a default sbt channel as follows (note that the PARMS value is vendor-specific) and then back up the database using these defaults:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt PARMS 'ENV=(NSR_SERVER=bksvr1)';
RMAN> BACKUP DATABASE;
```

RMAN preconfigures a DISK channel for you, so you can make disk backups using automatic channels without performing any configuration whatsoever.

The other method is to allocate channels manually within a RUN command. For example, this command allocates multiple disk channels and then backs up the database and archived redo logs:

```
RMAN> RUN
{
  ALLOCATE CHANNEL ch1 DEVICE TYPE DISK;
  ALLOCATE CHANNEL ch2 DEVICE TYPE DISK;
  ALLOCATE CHANNEL ch3 DEVICE TYPE DISK;
  BACKUP DATABASE PLUS ARCHIVELOG;
}
```

The following example manually allocates an sbt channel (with a vendor-specific PARMS value) and backs up a datafile copy:

```
RMAN> RUN
{
  ALLOCATE CHANNEL ch1 DEVICE TYPE sbt PARMS 'ENV=(NSR_SERVER=bksvr1)';
  BACKUP DATAFILECOPY '/tmp/system01.dbf';
}
```

For the most part, the procedures in this chapter assume that you have configured automatic channels.

Configuring the Default Backup Type for Disk

When backing up to disk, it is recommended to create image copies, rather than backup sets. Some features of RMAN backups, such as incrementally updated backups, require the use of image copies. Also, image copy backups are more convenient to use in some restore and recovery scenarios. However, by default, the BACKUP command creates backups as backup sets, when backing up to disk as well as to tape. (Backups to tape must be stored as backup sets.)

To configure RMAN to create image copies by default when backing up to disk, use the following command:

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY;
```

To return RMAN to its default behavior of producing backup sets, use the following command:

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE CLEAR;
```

Duplexing Backup Sets

It is safer to make multiple copies of backups to protect against disaster, media damage, or human error. RMAN can make up to four copies of a backup set simultaneously, each an exact duplicate of the others. A copy of a backup set is a copy of each backup piece in the backup set, with each copy getting a unique copy number (for example, `0tcm8u2s_1_1` and `0tcm8u2s_1_2`).

In most cases, the easiest method is to use `BACKUP... COPIES` or `CONFIGURE ... BACKUP COPIES` to duplex backup sets. There is little value in creating multiple copies on the same physical media. For `DISK` channels, specify multiple values in the `FORMAT` option to direct the multiple copies to different physical disks. For `sbt` channels, if you use a media manager that supports Version 2 of the SBT API, then the media manager will automatically put each copy onto a separate medium (for example, a separate tape).

Note that it is not possible to duplex backup sets to the flash recovery area, and that duplexing only applies to backup sets, not image copies. It is an error to specify the `BACKUP... COPIES` when creating image copy backups, and the `CONFIGURE... BACKUP COPIES` setting is ignored for image copy backups.

Duplexing Backup Sets with CONFIGURE BACKUP COPIES

The `CONFIGURE ... BACKUP COPIES` command specifies the number of identical backup sets that you want to create on the specified device type. This setting applies

to all backups except control file autobackups (because the autobackup of a control file always produces one copy) and backup sets when backed up with the BACKUP BACKUPSET command. You must have automatic channels configured.

To duplex a backup with CONFIGURE BACKUP COPIES:

1. Configure the number of copies on the desired device type for datafiles and archived redo logs on the desired device types. This example configures duplexing for datafiles and archived logs on tape as well as duplexing for datafiles (but not archived logs) on disk:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/save1/%U', '/save2/%U';
RMAN> CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE sbt TO 2;
RMAN> CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE sbt TO 2;
RMAN> CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 2;
```

2. Execute the BACKUP command. The following command backs up the database and archived logs to tape, making two copies of each datafile and archived redo log:

```
RMAN> BACKUP DATABASE PLUS ARCHIVELOG; # uses default sbt channel
```

   Because of the configured formats for the disk channel, the following command backs up the database to disk, placing one copy of the backupsets produced in the `/save1` directory and the other in the `/save2` directory:

```
RMAN> BACKUP DEVICE TYPE DISK AS COPY DATABASE;
```

3. Issue a LIST BACKUP command to see a listing of backup sets and pieces . For example, enter:

```
RMAN> LIST BACKUP SUMMARY;
```

   The #Copies column shows the number of backupsets, which may have been produced by duplexing or by multiple backup commands.

Duplexing Backupsets with BACKUP... COPIES

The COPIES option of the BACKUP command overrides every other COPIES or DUPLEX setting to control duplexing of backupsets.

To duplex a backup with BACKUP COPIES:

1. Specify the number of identical copies with the COPIES option of the BACKUP command. For example, run the following to make three copies of each backup set in the default DISK location:

```
RMAN> BACKUP AS BACKUPSET DEVICE TYPE DISK
  COPIES 3
  INCREMENTAL LEVEL 0
  DATABASE;
```

   Because you specified COPIES on the BACKUP command, RMAN makes three backupsets of each datafile regardless of the CONFIGURE DATAFILE COPIES setting.

2. Issue a LIST BACKUP command to see a listing of backup sets and pieces (the #Copies column shows the number of copies, which may have been produced through duplexing or through multiple invocations of the BACKUP command). For example, enter:

```
RMAN> LIST BACKUP SUMMARY;
```

Making Split Mirror Backups with RMAN

Many sites keep an backup of the database stored on disk in case a failure occurs on the primary database or an incorrect user action such as a DROP TABLE requires point-in-time recovery. A datafile backup on disk simplifies the restore step of recovery, making recovery much quicker and more reliable.

> **Caution:** Never make backups, split mirror or otherwise, of online redo logs. Restoring online redo log backups can create two archived logs with the same sequence number but different contents. Also, it is best to use the BACKUP CONTROLFILE command rather than a split mirror to make control file backups.

One way of creating a datafile backup on disk is to use disk mirroring. For example, you can use the operating system to maintain three identical copies of each file in the database. In this configuration, you can split off a mirrored copy of the database to use as a backup.

RMAN does not automate the splitting of mirrors, but can make use of split mirrors in backup and recovery operations. For example, RMAN can treat a split mirror of a datafile as a datafile copy, and can also back up this copy to disk or tape.

The following procedure shows how to make a split mirror backup with the SUSPEND/RESUME functionality. The SUSPEND/RESUME feature is not required for split mirror backups in most cases, although it is necessary if your system requires the database cache to be free of dirty buffers before the volume can be split.

To make a split mirror backup of a tablespace by using SUSPEND/RESUME:

1. Start RMAN and then place the tablespaces that you want to back up into backup mode with the ALTER TABLESPACE . . . BEGIN BACKUP statement. (To place all tablespaces in backup mode, you can use ALTER DATABASE BEGIN BACKUP instead.)

   For example, to place tablespace users in backup mode, start RMAN and run the following commands:

   ```
   RMAN> CONNECT TARGET SYS/oracle@trgt
   RMAN> CONNECT CATALOG rman/cat@catdb
   RMAN> SQL 'ALTER TABLESPACE users BEGIN BACKUP';
   ```

2. Suspend the I/Os if your mirroring software or hardware requires it. For example, enter the following SQL statement:

   ```
   RMAN> SQL 'ALTER SYSTEM SUSPEND';
   ```

3. Split the mirrors for the underlying datafiles contained in these tablespaces.

4. Take the database out of the suspended state:

   ```
   RMAN> SQL 'ALTER SYSTEM RESUME';
   ```

5. Take the tablespaces out of backup mode. For example, enter:

   ```
   RMAN> SQL 'ALTER TABLESPACE users END BACKUP';
   ```

   You could also use ALTER DATABASE END BACKUP to take all tablespaces out of backup mode.

6. Start an RMAN session and then catalog the user-managed mirror copies as datafile copies with the CATALOG command. For example, enter:

   ```
   RMAN> CATALOG DATAFILECOPY '/dk2/oradata/trgt/users01.dbf'; # catalog split
   mirror
   ```

7. Back up the datafile copies. For example, assuming that you have configured automatic channels, run the BACKUP DATAFILECOPY command at the prompt:

```
RMAN> BACKUP DATAFILECOPY '/dk2/oradata/trgt/users01.dbf';
```

8. When you are ready to resilver the split mirror, first use the CHANGE ... UNCATALOG command to uncatalog the datafile copies you cataloged in step 6. For example, enter:

```
RMAN> CHANGE DATAFILECOPY '/dk2/oradata/trgt/users01.dbf' UNCATALOG;
```

9. Resilver the split mirror for the affected datafiles.

> **See Also:** *Oracle Database SQL Reference* for ALTER SYSTEM SUSPEND syntax

Backing Up Backup Sets with RMAN

Use the BACKUP BACKUPSET command to back up backup sets rather than database files. This command is especially useful in the following scenarios:

- Ensuring that all backups exist both on disk and on tape
- Moving backups from disk to tape and then deallocating the space on disk

> **Note:** You cannot duplex backups when running BACKUP BACKUPSET. RMAN always makes one and only one copy on the specified media when performing BACKUP BACKUPSET.

To back up backup sets from disk to tape:

1. Assuming that you have configured an automatic sbt channel, issue the BACKUP BACKUPSET command at the RMAN prompt. This example backs up all disk backup sets to tape:

```
RMAN> BACKUP DEVICE TYPE sbt BACKUPSET ALL;
```

   This example backs up all disk backup sets to tape and then deletes the input disk backups:

```
RMAN> BACKUP DEVICE TYPE sbt BACKUPSET ALL DELETE INPUT;
```

2. Issue a LIST command to see a listing of backup sets and pieces.

Backing Up Image Copies with RMAN

Use the BACKUP COPY OF command to back up image copies of datafiles, control files, and archived logs. The output of the BACKUP command can be either backup sets or image copies, so you can generate backup sets from your image copies. This technique is useful when you want to back up a database backup on disk to tape, because all backups to tape must be backup sets. You can use the MAXSETSIZE parameter of the BACKUP command to set a maximum size for each backup set.

To back up image copies from disk to tape:

1. Assuming that you have configured an automatic sbt channel, issue the BACKUP COPY OF command at the RMAN prompt. This example backs up the latest image copy of the database to tape:

   ```
   RMAN> BACKUP DEVICE TYPE sbt COPY OF DATABASE;
   ```

   This example backs up the latest image copy backup of a database in backup sets on tape, and then deletes the input disk backups:

   ```
   RMAN> BACKUP DEVICE TYPE sbt COPY OF DATABASE DELETE INPUT;
   ```

2. Issue a LIST command to see a listing of backup sets and pieces.

When backing up your image copies, identifying the image copy to back up is easier if you provide tags for your backups. Image copies of datafiles and archived redo logs have associated tags (if you do not provide one, one is automatically generated). The tag of an image copy is inherited by default when the image copy is backed up as a new image copy. You can also specify your own tag.

To take advantage of the tags associated with your image copy backups, use the WITH TAG selector. As explained previously, the tag of the image copy being backed up will also be assigned to the new backup. When multiple image copies have the same tag, the most recent image copy of a file with the specified tag will be backed up.

Restarting and Optimizing RMAN Backups

RMAN supports two distinct features by which it can back up only those files that require backups: **restartable backups** and **backup optimization**.

With the restartable backup feature, RMAN backs up only those files that were not backed up after a specified date. For example, by specifying the NOT BACKED UP SINCE TIME clause, you can direct RMAN to back up only those files that were not backed up within the last day.

With backup optimization, the BACKUP command skips the backup of a file if the identical file has already been backed up to the allocated device type. To override this behavior and back up all files whether or not they have changed, specify the FORCE option on the BACKUP command. To enable or disable backup optimization, specify ON or OFF on the CONFIGURE BACKUP OPTIMIZATION command.

Additionally, BACKUP . . . PLUS ARCHIVELOG can archive unarchived online logs as well as back up archived logs.

> **See Also:** "Backup Optimization" on page 2-49 for a conceptual overview of optimization, and "Restartable Backups" on page 2-54 for a conceptual overview of restartable backups

Backing Up Files Using Backup Optimization

For backup optimization to be enabled, you must CONFIGURE BACKUP OPTIMIZATION to ON. Backup optimization is OFF by default.

To use backup optimization with a backup operation:

1. If you have not already enabled backup optimization, then enable it by running the CONFIGURE BACKUP OPTIMIZATION command. For example, enter:

   ```
   RMAN> CONFIGURE BACKUP OPTIMIZATION ON;
   ```

2. Back up the desired files. The following example backs up to an sbt device any archived redo logs that either have not been already backed up, or where the existing backups do not satisfy the current duplexing setting:

   ```
   RMAN> BACKUP DEVICE TYPE sbt ARCHIVELOG ALL;
   ```

   RMAN does not signal an error when it skips backing up files due to backup optimization.

   > **See Also:** "Backup Optimization" on page 2-49 for a conceptual overview of optimization and backup retention policies

Restarting a Backup After It Partially Completes

Use the SINCE TIME parameter of the BACKUP command to specify a date after which a new backup is required. If you do not specify the SINCE parameter, then RMAN only backs up files that have never been backed up.

To only back up files that were not backed up after a specified date:

Specify a valid date in the SINCE TIME parameter. For example, this command uses the default configured channel to back up all database files and archived redo logs that have not been backed up in the last two weeks:

```
RMAN> BACKUP NOT BACKED UP SINCE TIME 'SYSDATE-14'
          DATABASE PLUS ARCHIVELOG;
```

Validating Backups with RMAN

You can use the VALIDATE keyword of the BACKUP command to do the following:

- Check datafiles for physical and logical corruption
- Confirm that all database files exist and are in the correct locations

RMAN does not actually produce backup sets, but rather reads the specified files in their entirety, to determine whether they can be backed up and are not corrupted. In this sense, the BACKUP VALIDATE command is similar to the RESTORE VALIDATE command, except for backups rather than restore jobs.

If the backup validation discovers corrupt blocks, then RMAN updates the V$DATABASE_BLOCK_CORRUPTION view with rows describing the corruptions. After a corrupt block is repaired, the row identifying this block is deleted from the view.

For example, you can validate that all database files and archived redo logs can be backed up by running a command as follows:

```
RMAN> BACKUP VALIDATE DATABASE ARCHIVELOG ALL;
```

This form of the command would check for physical corruption. To check for logical corruption,

```
RMAN> BACKUP VALIDATE CHECK LOGICAL DATABASE ARCHIVELOG ALL;
```

RMAN displays the same output that it would if it were really backing up the files. If RMAN cannot validate the backup of one or more of the files, then it displays an error message. For example, RMAN may show output similar to the following:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of backup command at 08/29/2001 14:33:47
ORA-19625: error identifying file /oracle/oradata/trgt/arch/archive1_6.dbf
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
```

```
Additional information: 3
```

You cannot use the MAXCORRUPT or PROXY parameters with the VALIDATE option.

See Also:

- *Oracle Database Recovery Manager Reference* for BACKUP syntax
- "Recovering Blocks Listed in V$DATABASE_BLOCK_CORRUPTION" on page 8-23 to learn how to repair corrupt blocks discovered by BACKUP . . . VALIDATE

RMAN Backup Examples

This section contains these topics:

Specifying the Device Type on the BACKUP Command: Example

Assume that you configure an automatic sbt channel as follows:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1; # configure device
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt PARMS='...'; # configure options for
channels
RMAN> CONFIGURE DEFAULT DEVICE TYPE to sbt; # set default device type
```

Assume that you want to back up the database to disk and use the default configured DISK channel. You can specify that the BACKUP command should use a DISK channel as follows:

```
RMAN> BACKUP DEVICE TYPE DISK DATABASE;
```

To back up the database to the sbt device run this command:

```
RMAN> BACKUP DATABASE;
```

Skipping Tablespaces when Backing Up a Database: Example

The following example assumes that the database is running in ARCHIVELOG mode and that you have an automatic sbt channel configured as follows:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt PARMS='ENV=(NSR_DATA_VOLUME_
POOL=BackupPool)';
```

To back up the database while skipping offline and read-only tablespaces, you can run the following command:

```
RMAN> BACKUP DATABASE
        SKIP READONLY
        SKIP OFFLINE;
```

You need to back up a read-only tablespace only once after it has been made read-only. You can use the SKIP READONLY option to skip read-only datafiles. If you use the SKIP OFFLINE option, then the BACKUP command does not attempt to access offline datafiles. Use this option if the offline datafiles are not available.

Another way to persistently skip tablespaces across RMAN sessions is to issue the CONFIGURE EXCLUDE command for each tablespace that you always want to skip. For example, you may always want to skip the example tablespace, which has been made read-only. You can then issue:

```
RMAN> CONFIGURE EXCLUDE FOR TABLESPACE example;
```

Then, whenever you run BACKUP DATABASE, RMAN skips this tablespace. You do not have to specify a SKIP clause on the BACKUP command. You can override this behavior and include the example tablespace as follows:

```
RMAN> BACKUP DATABASE NOEXCLUDE;
```

Restarting a Backup: Example

Assume that you back up the database and archived logs every night to tape by running this command:

```
RMAN> BACKUP
  MAXSETSIZE 10G
  DATABASE PLUS ARCHIVELOG;
```

The preceding command sets an upper limit to the size of each backup set so that RMAN produces multiple backup sets. Assume that the media management device fails halfway through the backup and is then restarted. The next day you discover that only half the backup sets completed. In this case, you can run this command in the evening:

```
RMAN> BACKUP
  # Note that the NOT BACKED UP SINCE clause should be placed immediately after
the BACKUP
  # keyword or after each individual backupSpec clause
  NOT BACKED UP SINCE TIME 'SYSDATE-1'
  MAXSETSIZE 10M
  DATABASE PLUS ARCHIVELOG;
```

RMAN backs up only files that were not backed up during in the previous 24 hours. When RMAN finds out that particular file is already backed up it displays output similar to the following:

```
RMAN-06501: skipping datafile 1; already backed up on NOV 02 2003 18:10:00
RMAN-06501: skipping datafile 2; already backed up on NOV 02 2003 18:09:45
RMAN-06501: skipping datafile 3; already backed up on NOV 02 2003 18:09:45
```

Spreading a Backup Across Multiple Disk Drives: Example

Typically, you do not need to specify a format when backing up to tape because the default %U variable generates a unique filename for tape backups. When backing up to disk, however, you can specify a format if you need to spread the backup across several drives for improved performance. In this case, allocate one DISK channel for

each disk drive and specify the format string on the ALLOCATE CHANNEL command so that the filenames are on different disks. For example, issue:

```
RUN
{
  ALLOCATE CHANNEL disk1 DEVICE TYPE DISK FORMAT '/disk1/%d_backups/%U';
  ALLOCATE CHANNEL disk2 DEVICE TYPE DISK FORMAT '/disk2/%d_backups/%U';
  ALLOCATE CHANNEL disk3 DEVICE TYPE DISK FORMAT '/disk3/%d_backups/%U';
  BACKUP AS COPY DATABASE;
}
```

You can accomplish the same result by configuring automatic channels as follows:

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
CONFIGURE DEFAULT DEVICE TYPE TO DISK;
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '/disk1/%d_backups/%U';
CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT '/disk2/%d_backups/%U';
CONFIGURE CHANNEL 3 DEVICE TYPE DISK FORMAT '/disk3/%d_backups/%U';
BACKUP AS COPY DATABASE;
```

If you specify a nonexistent directory, RMAN displays output such as the following:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of backup command on ORA_DISK_1 channel at 08/29/2001
            14:36:04
ORA-19504: failed to create file "/nosuchdisk/0cd2momi_1_1"
ORA-27040: skgfrcre: create error, unable to create file
SVR4 Error: 2: No such file or directory
```

Backing Up a Large Database to Multiple File Systems: Example

In this scenario, you have a 35 GB database that you want to back up to disk. Because RMAN can only write one backup piece on a raw disk device, you decide to spread the backup across four file systems. You decide to make each backup set roughly the same size: 10 GB. You want each backup piece to be no more than 2 GB so that each backup set contains up to five backup pieces.

You decide to use the FORMAT parameter of the CONFIGURE CHANNEL command so that each channel will write to a different file system. You use conversion variables to guarantee unique names for the backup pieces. For example, the following commands configure four channels, configures their formats so that they write to the four file systems (/fs1, /fs2, /fs3, /fs4) and groups the datafiles so that each backup set is about the same size.

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 4;
RMAN> CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT='/fs1/%U' MAXPIECESIZE 2G;
RMAN> CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT='/fs2/%U' MAXPIECESIZE 2G;
RMAN> CONFIGURE CHANNEL 3 DEVICE TYPE DISK FORMAT='/fs3/%U' MAXPIECESIZE 2G;
RMAN> CONFIGURE CHANNEL 4 DEVICE TYPE DISK FORMAT='/fs4/%U' MAXPIECESIZE 2G;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO DISK;
```

Then, you can run this command every night to generate four backup sets, each in a different directory and each approximately the same size:

```
RMAN> BACKUP AS BACKUPSET DATABASE;
```

You can also back up the backup sets from disk to four different tapes from a tape pool by setting `PARALLELISM=4` for the `sbt` device (and specifying the appropriate vendor-specific `PARMS` for the `sbt` channel), as in the following example:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 4;
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt PARMS='...';
RMAN> BACKUP DEVICE TYPE sbt BACKUPSET ALL DELETE INPUT;
```

Note that this example makes certain assumptions, such as no datafile is too large to fit into a backup piece, that there are at least four datafiles in the database, and so on. In extrapolating from this example you must take into account the specifics of your own circumstances.

Specifying the Size of Backup Sets: Example

When making backups, RMAN divides the total number of files requiring backups by the number of allocated channels to calculate the number of files to place in each backup set. Use the `MAXSETSIZE` parameter to override this calculation and specify how many files should go in each backup set.

The `MAXSETSIZE` parameter specifies a maximum size for a backup set in units of bytes (default), kilobytes, megabytes, or gigabytes. Thus, to limit a backup set to 305 MB, specify `MAXSETSIZE=305M`. RMAN attempts to limit all sets to this size.

You can use `MAXSETSIZE` to limit the size of backup sets so that the database is divided among more than one backup set. If you configure `MAXSETSIZE` so that you generate multiple backup sets, however, then if the backup fails partway through, you can use the restartable backup feature to back up only those files that were not backed up during the previous attempt. See "Restartable Backups" on page 2-54 for a conceptual overview of restartable backups.

The following example configures a tape device, then backs up archived redo logs to tape, limiting the size to 100 MB so that if the backup fails partway through, it can be restarted:

```
RMAN> CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> BACKUP MAXSETSIZE = 100M ARCHIVELOG ALL;
```

This example accomplishes the same result with CONFIGURE MAXSETSIZE:

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> CONFIGURE MAXSETSIZE = 100M;
RMAN> BACKUP ARCHIVELOG ALL;
```

Note that if you specify a MAXSETSIZE value that is too small to contain the biggest file that you are backing up (either the actual size of that file, or if binary compression is specified, then the size of tha tfile after compression), then RMAN displays an error stack such as the following:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of backup command at 11/03/03 14:40:33
RMAN-06182: archive log larger than MAXSETSIZE: thread 1 seq 1
            /oracle/oradata/trgt/arch/archive1_1.dbf
```

Limiting the Size of Backup Pieces: Example

Backup piece size is an issue in those situations where it exceeds the maximum file size of the file system or media management software. Use the MAXPIECESIZE parameter of the CONFIGURE CHANNEL or ALLOCATE CHANNEL command to limit the size of backup pieces.

For example, to limit the backup file size to 2GB or less, you can configure the automatic DISK channel as follows and then run BACKUP DATABASE:

```
# max file size for backup pieces is 2GB
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 2GB;
RMAN> BACKUP DATABASE;
```

Note that in version 2.0 of the media management API, media management vendors can specify the maximum size of a backup piece that can be written to their media manager. RMAN will respect this limit regardless of the settings you configure for MAXPIECESIZE.

Backing Up Archived Redo Logs in a Failover Scenario: Example

Assume that you set your initialization parameters so that you archive to the following local destinations:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arch/'
LOG_ARCHIVE_DEST_2 = 'LOCATION=/disk2/arch/'
LOG_ARCHIVE_DEST_3 = 'LOCATION=/disk3/arch/'
```

Each directory contains the same set of logs, starting with log sequence 1 and ending at log sequence 400. Unknown to you, a user inadvertently deletes logs 300 through 400 from `/disk1/arch` and logs 350 through 400 from `/disk2/arch`. You run this backup command:

```
RMAN> BACKUP ARCHIVELOG
  FROM SEQUENCE 288 UNTIL SEQUENCE 388
  THREAD 1
  DELETE INPUT;
```

RMAN begins backing up logs starting with log sequence 288. If the copy of log 300 that was deleted from `/disk1/arch` is the one that RMAN attempts to back up, then RMAN checks the repository to determine whether other copies of this log sequence exist, and backs up the log in either `/disk2/arch` or `/disk3/arch`. Hence, because a copy of each log in sequence 288 through 388 is located in at least one of the three directories, RMAN can back up all the specified logs.

Backing Up Archived Logs Needed to Recover an Online Backup: Example

Assume that you back up database `trgt` while it is open. You want to back up only those archived redo logs required to recover this online backup.

The recommended solution to this problem is to add the `PLUS ARCHIVELOG` clause to your database backup command, as shown here:

```
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```

See *Oracle Database Backup and Recovery Basics* and *Oracle Database Recovery Manager Reference* for details on using `BACKUP... PLUS ARCHIVELOG`.

Prior to Oracle Database Release 10*g* , however, you could manually determine which archived logs are required and back them up, using the following procedure.

To determine the archived logs needed for recovery of an online backup:

1. Start SQL*Plus and archive all unarchived logs, including the current log:

   ```
   SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
   ```

2. Query V$LOG to determine the log sequence number of the current redo log, as in the following example (which includes output):

```
SQL> SELECT SEQUENCE# FROM V$LOG WHERE STATUS = 'CURRENT';

 SEQUENCE#
----------
      9100
```

3. Start RMAN and make an online backup of the database. For example, enter:

```
RMAN> BACKUP DATABASE;
```

4. Archive all unarchived logs, including the current log:

```
RMAN> SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
```

5. In SQL*Plus, query V$LOG to determine the log sequence number of the current redo log:

```
SQL> SELECT SEQUENCE# FROM V$LOG WHERE STATUS = 'CURRENT';

 SEQUENCE#
----------
      9112
```

6. Back up the logs beginning with the first sequence number that you queried, and ending with the last sequence number minus 1. The log before the current log is the most recent archived log. For example, if the first query returned 9100, then start at 9100. If the second query returned 9112, then end at 9111.

   For example, issue the following to back up the necessary archived logs:

```
RMAN> BACKUP ARCHIVELOG FROM SEQUENCE 9100 UNTIL SEQUENCE 9111;
```

Backing Up and Deleting Multiple Copies of an Archived Redo Log: Example

In this scenario, you set initialization parameters so that you automatically archive redo logs to two directories: `?/oradata/trgt/arch/dest_1` and `?/oradata/trgt/arch/dest_2`. Therefore, you have two identical copies of the archived redo log for each log sequence number. You decide to back up each copy of the archived redo logs and then delete the originals. (Note that the degree of backup duplexing configured or specified in the BACKUP command determines the number of output files, independent of the number of input files. See "Duplexing Backup Sets" on page 7-3 for details.)

The easiest solution in this case is to use the DELETE ALL INPUT option means that RMAN deletes all logs that match the ARCHIVELOG criteria. Hence, it can remove all logs from both ?/oradata/trgt/arch/dest_1 and ?/oradata/trgt/arch/dest_2.

For example, run the following command to back up all logs that could be used to recover from a point 10 days ago, and then delete all logs within the specified time range from disk:

```
RMAN> BACKUP DEVICE TYPE sbt
  ARCHIVELOG ALL FROM TIME 'SYSDATE-10'
  DELETE ALL INPUT;
```

Performing Differential Incremental Backups: Example

A differential incremental backup contains only blocks that have been changed since the most recent backup at the same level or lower. The first incremental backup must be a level 0 backup that contains all used blocks. The following is a level 0 base backup:

```
RMAN> BACKUP INCREMENTAL LEVEL 0 DATABASE;
```

An incremental backup at level 1 will contain all blocks changed since the most recent level 1 backup. If no previous level 1 backup is available, then RMAN copies all blocks changed since the base level 0 backup. The following is a level 1 backup of the database:

```
RMAN> BACKUP INCREMENTAL LEVEL 1 DATABASE;
```

You can perform incremental backups in NOARCHIVELOG mode, but the backups must be consistent. Hence, you cannot take online incremental backups.

Performing Cumulative Incremental Backups: Example

A cumulative incremental backup at level 1 contains only blocks that have been changed since the most recent backup at level 0. Cumulative backups require more storage space than differential backups, but they are preferable during a restore operation because only one backup for a given level is needed. Note that the first incremental backup must be a level 0 backup that contains all used blocks.

In contrast to a cumulative backup, a differential backup at level 1 will determine which level 0 *or* level 1 backup occurred most recently and copy all blocks changed since this backup.

```
BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE; # blocks changed since level 0
```

Determining How Channels Distribute a Backup Workload: Example

When you create multiple backup sets and allocate multiple channels, RMAN automatically writes multiple backup sets in parallel. The allocated server sessions share the work of backing up the specified datafiles, control files, and archived redo logs. Note that you cannot stripe a single backup set across multiple channels.

RMAN automatically assigns a backup set to a device. You can use the CHANNEL parameter so that RMAN writes all backup sets for a *backupSpec* to a specific channel.

For example, this example parallelizes the backup operation by specifying which channels RMAN should back up to disk and which to sbt:

```
RMAN> RUN
{
  ALLOCATE CHANNEL ch1 DEVICE TYPE DISK FORMAT = '/backup/df/%U';
  ALLOCATE CHANNEL ch2 DEVICE TYPE DISK FORMAT = '/backup/cf/%U';
  ALLOCATE CHANNEL ch3 DEVICE TYPE sbt;
  BACKUP AS BACKUPSET # all output files are in backup sets
    # channel ch1 backs up datafiles to /backup/df directory
    DATAFILE 1,2,3,4
    CHANNEL ch1
    # channel ch2 backs up control file copy to /backup/cf directory
    CONTROLFILECOPY '/tmp/control01.ctl'
    CHANNEL ch2;
   BACKUP AS BACKUPSET
    # channel ch3 backs up archived redo logs to tape
    ARCHIVELOG FROM TIME 'SYSDATE-14'
    CHANNEL ch3;
}
```

You cannot back up to DISK and sbt at the same time using automatic channels: you must manually allocate them.

Backing Up in NOARCHIVELOG Mode: Example

This script puts the database into the correct mode for a consistent, whole database backup and then backs up the database. The script performs a shutdown, startup, shutdown, and then startup again before creating multiple copies of the backup:

```
# Shut down database cleanly with immediate option. This type of shutdown lets
# current calls to the database complete, but prevents further logons or calls.
# If the database is not up, you receive a message saying so but RMAN will not
# treat this situation as an error.
SHUTDOWN IMMEDIATE;
```

```
# Start up the database in case it suffered instance failure or was
# closed with SHUTDOWN ABORT before starting this script.
# The script performs instance recovery if
# needed. Oracle uses the default init.ora file. Alternatively, use this form:
# STARTUP FORCE DBA pfile=filename.
# Use the DBA option because you are going to shut down again
# and do not want to let users in during the short interval. Use the FORCE
# option because it cannot hurt and might help in certain situations.
STARTUP FORCE DBA;
SHUTDOWN IMMEDIATE;

# The database is cleanly closed and ready for a consistent backup. RMAN
# requires that the database be started and mounted to perform a backup.
RMAN> STARTUP MOUNT;

# this example uses automatic channels to make the backup
BACKUP
  COPIES 2
  INCREMENTAL LEVEL 0
  MAXSETSIZE 10M
  DATABASE;

# Now that the backup is complete, open the database.
ALTER DATABASE OPEN;
```

You can skip tablespaces, but any skipped tablespace that has not been offline or read-only since its last backup will be lost if the database has to be restored from a backup. When backing up to disk, make sure that the destination file system has enough free space.

Cataloging User-Managed Datafile Copies: Example

You can use operating system utilities to make copies of datafiles and then catalog them in the recovery catalog.

If the database is open and the datafile is online, then issue ALTER TABLESPACE . . . BEGIN BACKUP to put the datafile in backup mode. (If you will be copying all datafiles, you can use ALTER DATABASE BEGIN BACKUP to put all tablespaces in backup mode, instead of issuing individual commands for each tablespace.) Then copy the files using native operating system commands. Once the copy is completed, use ALTER TABLESPACE... END BACKUP to take the datafile out of backup mode (or use ALTER DATABASE END BACKUP to take all tablespaces in

backup mode out of backup mode). Finally, catalog the resulting datafile copy using the RMAN CATALOG DATAFILECOPY command.

For example, if you created a copy of your datafile /oracle/oradata/users01.dbf into file /tmp/users01.dbf, this command will add /tmp/users01.dbf to the catalog:

```
CATALOG DATAFILECOPY '/tmp/users01.dbf';
```

If you try to catalog a datafile copy from a database other than the connected target database, then RMAN issues an error such as the following:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of catalog command on default channel at 08/29/2001 14:44:34
ORA-19563: datafile copy header validation failed for file /tmp/tools01.dbf
```

Keeping a Long-Term Backup: Example

If you configure a retention policy, then you may want to exclude specified backups from this policy. For example, you may want to archive a consistent backup of the database once a year to serve as a historical record. This long-term backups does not function as a backup that you may perform recovery on, but an archived snapshot of data at a particular time.

To exempt a backup from the retention policy, specify the KEEP option on the BACKUP command. You can also specify LOGS or NOLOGS to indicate whether RMAN should save archived logs for possible recovery of this backup. If you specify NOLOGS, then the backup must be consistent.

This example keeps the backup of the database indefinitely and does not save archived logs needed to recover it:

```
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;  # put database in consistent state
RMAN> BACKUP DATABASE KEEP FOREVER NOLOGS
              TAG 'db_archive_1'; # make long-term consistent backup

# mark backup as unavailable in the repository so that RMAN does not attempt to
# restore it unless explicitly specified on the RESTORE command
RMAN> CHANGE BACKUP TAG 'db_archive_1' UNAVAILABLE;
RMAN> SQL 'ALTER DATABASE OPEN';
```

Optimizing Backups: Examples

Run the CONFIGURE BACKUP OPTIMIZATION command to enable backup optimization. When specific conditions are met (described in "Backup Optimization Algorithm" on page 2-49), RMAN skips backups of files that are identical to files that are already backed up.

Assume that you configure optimization and a retention policy as follows:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 4 DAYS;
```

Optimizing a Database Backup: Example

Then, you run this command every night to back up the database to tape:

```
BACKUP DATABASE;
```

Because backup optimization is configured, RMAN skips backups of offline and read-only datafiles only if the most recent backups were made on or after the earliest point in the recovery window. RMAN does not skip backups when the most recent backups are older than the window. For example, optimization ensures you do not end up with a new backup of the read-only datafile ?/oradata/trgt/history01.dbf every night, so long as one backup set containing this file exists within the recovery window.

For example, if the most recent backup of the datafiles was on Sunday, and the point of recoverability (that is, the earliest date in the recovery window) is on Saturday, then RMAN skips the datafiles when you run the Wednesday backup. On Friday, the point of recoverability is now Monday, so the Sunday backup is now outside the window. Hence, the Friday backup does not skip the datafiles.

Optimizing a Daily Archived Log Backup to a Single Tape: Example

Assume that you want to back up all the archived logs every night. However, you do not want to have multiple copies of each log sequence number. So, you configure backup optimization to ON, then run this command in a script every night at 1 a.m.:

```
BACKUP DEVICE TYPE sbt ARCHIVELOG ALL;
```

RMAN skips all logs except those produced in the last 24 hours. In this way, you keep only one copy of each archived log on tape.

Optimizing a Daily Archived Log Backup to Multiple Tapes: Example

In this example, you back up logs that are not already on tape to one tape pool, then back up the same logs to a second tape pool. Finally, you delete old logs.

For the first step, perform the one-time configuration:

```
# configure backup optimization
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

Then, run the following script at the same time every night to back up the logs generated during the previous day to two separate tape pools:

```
# The following command backs up just the logs that are not on tape. The
# first copies are saved to the tapes from the pool "archivelog_pool_1"
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt
    PARMS='NSR_DATA_VOLUME_POOL=ARCHIVELOG_POOL_1';
  BACKUP ARCHIVELOG ALL;
}
# Make one more copy of the archived logs and save them to tapes from a
# different pool
RUN
{
  ALLOCATE CHANNEL c2 DEVICE TYPE sbt
    PARMS='NSR_DATA_VOLUME_POOL=ARCHIVELOG_POOL_2';
  BACKUP ARCHIVELOG
    FROM TIME 'SYSDATE-1'
    UNTIL TIME 'SYSDATE'; # specify UNTIL so RMAN does not archive current log
}
# Delete old logs - for example, delete logs created within the last week.
DELETE ARCHIVELOG ALL COMPLETED AFTER 'SYSDATE-7';
```

Creating a Weekly Secondary Backup of Archived Logs: Example

Assume a more sophisticated scenario in which your goal is to back up the archived logs to tape every day. However, you are worried about tape failure, so you want to ensure that you have more than copy of each log sequence number on an separate tape before you perform your weekly deletion of logs from disk.

First, perform a one-time configuration:

```
# configure backup optimization
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
```

```
CONFIGURE default DEVICE TYPE TO sbt;
# configure a default channel that sends backups to tape pool "first_copy"
CONFIGURE CHANNEL DEVICE TYPE sbt PARMS='ENV=(NSR_DATA_VOLUME_POOL=first_copy);
```

Because you have optimization enabled, you can run the following command every evening to back up all archived logs to the "first_copy" pool that have not already been backed up:

```
BACKUP ARCHIVELOG ALL TAG first_copy;
```

Every Friday evening you create an additional backup of all archived logs in a different tape pool. Also, at the end of the backup, you want to delete all archived logs that already have at least two copies on tape. So you run the following script:

```
RUN
{
  # manually allocate a channel, in order to specify that the backup run by this
  # channel should go to both pools "first_copy" and "second_copy"
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt
      PARMS='ENV=(NSR_DATA_VOLUME_POOL=second_copy)';
  ALLOCATE CHANNEL c2 DEVICE TYPE sbt
      PARMS='ENV=(NSR_DATA_VOLUME_POOL=first_copy)';
  BACKUP CHANNEL C1 ARCHIVELOG UNTIL TIME 'SYSDATE'
    NOT BACKED UP 2 TIMES # back up only logs without 2 backups on tape
    TAG SECOND_COPY;
  BACKUP CHANNEL C2 ARCHIVELOG UNTIL TIME 'SYSDATE'
    NOT BACKED UP 2 TIMES # back up only logs without 2 backups on tape
    TAG FIRST_COPY;
}

# now delete from disk all logs that have been backed up to tape at least twice
DELETE ARCHIVELOG ALL
  BACKED UP 2 TIMES TO DEVICE TYPE sbt;
```

The Friday script creates a second copy of all archived logs in the "second_copy" tape pool. After the backup, you can send the tape from the pool "second_copy" to secure storage. You should use this tape backup only if the primary tape from pool "first_copy" is damaged. Because the secondary tape is in a secure place, you do not want RMAN to use it for recovery, so you can mark the backup as unavailable:

```
CHANGE BACKUP OF ARCHIVELOG TAG SECOND_COPY UNAVAILABLE;
```

Handling Errors During Backups: Example

By default a checksum is calculated for every block read from a datafile and stored in the backup or image copy. If you use the NOCHECKSUM option, then checksums are not calculated. If the block already contains a checksum, however, then the checksum is validated and stored in the backup. If the validation fails, then the block is marked corrupt in the backup.

The SET MAXCORRUPT FOR DATAFILE command sets how many corrupt blocks in a datafile that BACKUP will tolerate. If a datafile has more corrupt blocks than specified by the MAXCORRUPT parameter, the command terminates. If you specify the CHECK LOGICAL option, RMAN checks for logical and physical corruption.

By default, the BACKUP command terminates when it cannot access a datafile. You can specify parameters to prevent termination, as listed in the following table.

If you specify the option ...	Then RMAN skips...
SKIP INACCESSIBLE	Inaccessible datafiles. A datafile is only considered inaccessible if it cannot be read. Some offline datafiles can still be read because they exist on disk. Others have been deleted or moved and so cannot be read, making them inaccessible.
SKIP OFFLINE	Offline datafiles.
SKIP READONLY	Datafiles in read-only tablespaces.

The following example uses an automatic channel to back up the database, and sets the corruption level for the datafile in the SYSTEM tablespace so that up to 10 errors will be accepted:

```
RMAN> RUN
{
  SET MAXCORRUPT FOR DATAFILE 1 TO 10;
  BACKUP DATABASE
    SKIP INACCESSIBLE
    SKIP READONLY
    SKIP OFFLINE;
}
```

8

Advanced RMAN Recovery Techniques

This chapter describes how to use Recovery Manager to perform restore and recovery operations. This chapter contains these topics:

Performing Database Point-In-Time Recovery

RMAN can perform recovery of the whole database to a specified past time, SCN, or log sequence number. This type of recovery is sometimes called **incomplete recovery** because it does not completely use all of the available redo. Incomplete recovery of the whole database is also called **database point-in-time recovery (DBPITR)**.

If you have enabled the collectionof flashback logs, you may be able to use Oracle Flashback Database instead of performing DBPITR. Flashback Database is generally faster and simpler to use, when it is available, because it does not require restoring a past backup. Depending upon your situation, you may also find one of the other Oracle flashback features can meet your data recovery need. See "Oracle Flashback Technology: Overview" on page 9-2 for more details about these alternatives before deciding whether to use DBPITR.

DBPITR requires restoring your database from an older backup, then performing media recovery until your specified target time, SCN or log sequence number. Note that because you need your archived redo log files to perform this process, you cannot perform database point-in-time recovery if you have been running your database in NOARCHIVELOG mode.

After database point-in-time recovery, you must open the database with the `RESETLOGS` option. Using the `RESETLOGS` option archives the current online redo logs, resets the log sequence to 1, and then gives the online redo logs a new time stamp and SCN. In this way, the database eliminates the possibility of corrupting datafiles by the application of obsolete archived redo logs.

You have to recover *all* datafiles: you cannot recover some datafiles before the `RESETLOGS` and others after the `RESETLOGS`.

The `OPEN RESETLOGS` operation will fail if a datafile is off-line, unless the datafile went offline normally or is read-only. You can bring files in read-only or offline normal tablespaces online after the `RESETLOGS` because they do not need any redo.

When performing DBPITR, consider using the `SET UNTIL` command to set the target time at the beginning of the process, rather than specifying the `UNTIL` clause on the `RESTORE` and `RECOVER` commands individually. `SET UNTIL` sets the desired time for any subsequent `RESTORE`, `SWITCH`, and `RECOVER` commands in the same `RUN` job.

Note that if you specify a `SET UNTIL` command after a `RESTORE` and before a `RECOVER`, you may not be able to recover the database to the point in time required because the restored files may already have time stamps more recent than the set

time. Hence, it is recommended that you specify the SET UNTIL command *before* the RESTORE command.

Performing Point-in-Time Recovery with a Current Control File

The database must be closed to perform database point-in-time recovery. If you are recovering to a time, then you should set the time format environment variables before invoking RMAN. The following are sample Globalization Support settings:

```
NLS_LANG = american_america.us7ascii
NLS_DATE_FORMAT="Mon DD YYYY HH24:MI:SS"
```

To recover the database until a specified time, SCN, or log sequence number:

1. After connecting to the target database and, optionally, the recovery catalog database, ensure that the database is mounted. If the database is open, shut it down and then mount it:

   ```
   SHUTDOWN IMMEDIATE;
   STARTUP MOUNT;
   ```

2. Determine the time, SCN, or log sequence that should end recovery. For example, if you discover that a user accidentally dropped a tablespace at 9:02 a.m., then you can recover to 9 a.m.—just before the drop occurred. You will lose all changes to the database made after that time.

   You can also examine the alert.log to find the SCN of an event and recover to a prior SCN. Alternatively, you can determine the log sequence number that contains the recovery termination SCN, and then recover through that log. For example, query V$LOG_HISTORY to view the logs that you have archived.

   ```
   RECID      STAMP      THREAD#    SEQUENCE#  FIRST_CHAN FIRST_TIM NEXT_CHANG
   ---------- ---------- ---------- ---------- ---------- --------- ----------
            1  344890611          1          1      20037 24-SEP-02      20043
            2  344890615          1          2      20043 24-SEP-02      20045
            3  344890618          1          3      20045 24-SEP-02      20046
   ```

3. Perform the following operations within a RUN command:

   a. Set the end recovery time, SCN, or log sequence. If specifying a time, then use the date format specified in the NLS_LANG and NLS_DATE_FORMAT environment variables.

   b. If automatic channels are *not* configured, then manually allocate one or more channels.

c. Restore and recover the database.

The following example performs an incomplete recovery until November 15 at 9 a.m.

```
RUN
{
  SET UNTIL TIME 'Nov 15 2002 09:00:00';
  # SET UNTIL SCN 1000;       # alternatively, specify SCN
  # SET UNTIL SEQUENCE 9923;  # alternatively, specify log sequence number
  RESTORE DATABASE;
  RECOVER DATABASE;
}
```

4. If recovery was successful, then open the database and reset the online logs:

```
ALTER DATABASE OPEN RESETLOGS;
```

Point-in-Time Recovery to a Previous Incarnation

RMAN can seamlessly restore and recover backups from previous incarnations to the current incarnation. To perform point-in-time recovery to a target time prior to the most recent RESETLOGS, however, you must run the RESET DATABASE command to reset the database to the incarnation current at the desired target time.

Assume the following situation:

- You run RMAN with a recovery catalog.
- You made a backup of target database `trgt` on October 2, 2002.
- You performed incomplete recovery on this database and opened it with the RESETLOGS option on October 10, 2002. A new database incarnation was created.

On October 25, you discover that you need crucial data that was dropped from the database at 8:00 a.m. on October 8, 2002. You decide to reset `trgt` to the prior incarnation, restore the October 2 backup, and recover to 7:55 a.m. on October 8.

Note: It is not possible to restore one datafile of a previous incarnation while the current database is in a different incarnation—you must restore the whole database.

To recover the database by means of a backup from the old incarnation:

1. Obtain the primary key of the prior incarnation with a LIST command:

```
# obtain primary key of old incarnation
LIST INCARNATION OF DATABASE trgt;

List of Database Incarnations
DB Key  Inc Key   DB Name   DB ID       STATUS     Reset SCN    Reset Time
------- -------   -------   ------      -------    ----------   ----------
1       2         TRGT      1224038686  PARENT     1            02-OCT-02
1       582       TRGT      1224038686  CURRENT    59727        10-OCT-02
```

2. Make sure the database is started but not mounted:

```
SHUTDOWN FORCE NOMOUNT
```

3. Reset the incarnation to the primary key that you just obtained:

```
# reset database to old incarnation
RESET DATABASE TO INCARNATION 2;
```

4. Recover the database, performing the following actions in the RUN command:

   - Set the end time for recovery to the time just before the loss of the data.
   - If automatic channels are not configured, then manually allocate one or more channels.
   - Restore the control file and mount it.
   - Restore and recover the database.

   For example, run the following commands:

```
RUN
{
  # set time to just before data was lost.
  SET UNTIL TIME 'Oct 8 2002 07:55:00';
  RESTORE CONTROLFILE; # FROM AUTOBACKUP not needed in catalog mode
  ALTER DATABASE MOUNT; # mount database after restoring control file
  RESTORE DATABASE;
  RECOVER DATABASE;
}
```

5. If recovery is successful, then reset the online redo logs:

```
# this command automatically resets the database so that this incarnation is
# the new incarnation
```

```
ALTER DATABASE OPEN RESETLOGS;
```

Performing Recovery with a Backup Control File

If all copies of the current control file are lost or damaged, then you must restore and mount a backup control file before you can perform recovery. There are two cases to consider:

- Performing Recovery with a Backup Control File and a Recovery Catalog
- Performing Recovery with a Backup Control File and No Recovery Catalog

The following notes and restrictions apply regardless of whether you use a recovery catalog:

- You must run the RECOVER command after restoring a backup control file, even if no datafiles have been restored.
- After restoring a backup control file, entries for tempfiles in locally-managed temporary tablespaces are removed. Hence, you must add new tempfiles to these tablespaces after you open with the RESETLOGS option. If you do not, then the database can display the following error when attempting to sort: `ORA-25153: Temporary Tablespace is Empty`.
- You must open the database with the RESETLOGS option after performing either complete *or* point-in-time recovery with a backup control file.
- If the online redo logs are inaccessible, then you must perform incomplete recovery to an SCN before the earliest SCN in the online redo logs. This limitation is necessary because RMAN does not back up online logs.
- RMAN automatically searches in specific locations for online and archived redo logs during recovery that are not recorded in the RMAN repository, and catalogs any that it finds. RMAN attempts to find a valid archived log in any of the current archiving destinations with the current log format. The current format is specified in the initialization parameter file used to start the instance (or all instances in a Real Application Clusters installation). Similarly, RMAN attempts to find the online redo logs by using the filenames as specified in the control file.

  If you changed the archiving destination or format during recovery, or if you added new online log members after the backup of the control file, then RMAN may not be able to automatically catalog a needed online or archived log. In this situation, RMAN reports errors similar to the following:

  ```
  RMAN-00571: ===========================================================
  ```

```
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of recover command at 08/29/2001 14:23:09
RMAN-06054: media recovery requesting unknown log: thread 1 scn 86945
```

In this case, you must use the CATALOG command to manually add the required logs to the repository so that recovery can proceed. The cataloging procedure is described in *Oracle Database Backup and Recovery Basics*.

Performing Recovery with a Backup Control File and a Recovery Catalog

If you use a recovery catalog and have a backup control file available, then this procedure does not differ substantially from a standard restore and recovery. The procedure in this section assumes that you are restoring the control file to its default location. If you must restore the control file to a new location, then refer to *Oracle Database Backup and Recovery Basics* for instructions.

When you perform a restore operation using a backup control file and you use a recovery catalog, RMAN automatically adjusts the control file to reflect the structure of the restored backup.

The following procedure assumes that you do *not* have more than one target database registered in the catalog with the same name. If multiple target databases *are* registered with the same name, then you must specify the DBID with the SET DBID command so that RMAN knows which control file to restore. The DBID is the unique numerical identifier for a database.

> **See Also:** "Performing Recovery with a Backup Control File and No Recovery Catalog" on page 8-8 to learn how to set the DBID

To recover the database with a backup control file and a recovery catalog:

1. After connecting to the target database and recovery catalog database, start the instance without mounting the database:

   ```
   STARTUP NOMOUNT
   ```

2. Restore the backup control file, then restore and recover the database. Do the following:

   a. Run the RESTORE CONTROLFILE command to restore the control file to all default locations specified in the CONTROL_FILES initialization parameter. To restore a control file from an older backup, you can run SET UNTIL or specify the UNTIL clause on the RESTORE CONTROLFILE command.

b. Mount the restored control file.

c. Optionally, run a SET UNTIL command for incomplete recovery. Note that you can also specify the UNTIL clause on the RESTORE and RECOVER commands.

d. Restore and recover the database

This example restores the control file to its default location, then restores and completely recovers the database:

```
RESTORE CONTROLFILE;
ALTER DATABASE MOUNT;
RESTORE DATABASE;
RECOVER DATABASE;
```

3. If recovery was successful, then open the database and reset the online logs:

```
ALTER DATABASE OPEN RESETLOGS;
```

4. If the database uses locally-managed temporary tablespaces, then add new tempfiles to these tablespaces. For example:

```
SQL "ALTER TABLESPACE temp
     ADD TEMPFILE ''?/oradata/trgt/temp01.dbf'' REUSE";
```

Performing Recovery with a Backup Control File and No Recovery Catalog

This section assumes that you have RMAN backups of the control file, but do not use a recovery catalog. Assuming that you enabled the control file autobackup feature for the target database, you can restore an autobackup of the control file. Because the autobackup uses a default format, RMAN can restore it even though it does not have a repository available that lists the available backups. You can restore the autobackup to the default or a new location. RMAN replicates the control file to all CONTROL_FILES locations automatically.

Note: If you know the backup piece name (for example, from the media manager or because the piece is on disk), then you can specify the piece name using the RESTORE CONTROLFILE FROM *'filename'* command. The server records the location of every autobackup in the alert log.

Because you are not connected to a recovery catalog, the control file must have a record of all needed backups. If any backups are not listed in the control file, then

RMAN cannot restore them. You can add backup pieces and image copies to the control file repository with the CATALOG command.

Because the repository is not available when you restore the control file, run the SET DBID command to identify the target database. You should only run the SET DBID command in the following specialized circumstances:

- You are not connected to a recovery catalog and want to restore the control file or server parameter file.
- You are connected to a recovery catalog and want to restore the control file, but the database name is not unique in the recovery catalog.
- The server parameter file is lost and you want to restore it.

To recover the database with an autobackup of the control file without a recovery catalog:

1. Start RMAN and connect to the target database. For example, run:

   ```
   CONNECT TARGET /
   ```

2. Start the target instance without mounting the database. For example:

   ```
   STARTUP NOMOUNT;
   ```

3. Set the database identifier for the target database with SET DBID. RMAN displays the DBID whenever you connect to the target. You can also obtain it by inspecting saved RMAN log files, querying the catalog, or looking at the filenames of control file autobackup. (refer to "Restoring When Multiple Databases in the Catalog Share the Same Name: Example" on page 8-25). For example, run:

   ```
   SET DBID 676549873;
   ```

4. Restore the autobackup control file, then perform recovery. Do the following:

   a. Optionally, specify the most recent backup time stamp that RMAN can use when searching for a control file autobackup to restore.

   b. If a nondefault format was used to create the control file, then specify a nondefault format for the restore of the control file.

   c. If the channel that created the control file autobackup was device type sbt, then you must allocate one or more sbt channels. Because no repository is available, you cannot use preconfigured channels. If the autobackup was created on a disk channel, however, then you do not need to manually allocate a channel.

d. Restore the autobackup of the control file, optionally setting the maximum number of days backward that RMAN can search (up to 366) and the initial sequence number that it should use in its search for the first day.

e. If you know that your control file contained information about configured channels that will be useful to you in the rest of the restore process, you can exit the RMAN client at this point, to clear manually allocated channels from step "c". If you then restart the RMAN client and mount the database those configured channels become available for your use in the rest of the restore and recovery process.

If you do not care about using configured channels from your control file, then you can simply mount the database at this point.

f. If the online logs are inaccessible, then restore and recover the database as described in "Performing Database Point-In-Time Recovery" on page 8-2. You must terminate recovery by setting the `UNTIL` clause to a time, log sequence, or SCN before the online redo logs. If the online logs are usable, then perform a complete recovery as described in *Oracle Database Backup and Recovery Basics*.

In this example, the online redo logs have been lost. This example limits the restore of the control file autobackup, then performs recovery of the database to log sequence 13243, which is the most recent archived log:

```
RUN
{
  # Optionally, set upper limit for eligible time stamps of control file
  # backups
  # SET UNTIL TIME '09/10/2000 13:45:00';
  # Specify a nondefault autobackup format only if required
  # SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK
  #   TO '?/oradata/%F.bck';
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS='...'; # allocate manually
  RESTORE CONTROLFILE FROM AUTOBACKUP
    MAXSEQ 100           # start at sequence 100 and count down
    MAXDAYS 180;         # start at UNTIL TIME and search back 6 months
  ALTER DATABASE MOUNT DATABASE;
}
# uses automatic channels configured in restored control file
RESTORE DATABASE UNTIL SEQUENCE 13243;
RECOVER DATABASE UNTIL SEQUENCE 13243; # recovers to latest archived log
```

5. If recovery was successful, then open the database and reset the online logs:

```
ALTER DATABASE OPEN RESETLOGS;
```

Restoring the Database to a New Host

Various scenarios are possible when restoring a database to a new host. For example, you may want to:

- Create a duplicate version of the production database for testing or other purposes, while retaining the production database on the original host.
- Test the restore of the production database to a new host, while retaining the production database on the original host.
- Move the location of the production database to a new host (possibly because the original host has failed).

To create a duplicate database for testing while maintaining the original database, use the `DUPLICATE` command instead of the `RESTORE` command (refer to "Duplicating a Database with Recovery Manager" on page 11-1). RMAN automatically creates a unique database identifier for the duplicate database. This chapter covers the use of the `RESTORE` command only.

To test the restore of a database to a new host or to move the database to a new host, run the `RESTORE` command. If you perform a test restore only, then you should do the following to prevent overwriting the target records in the recovery catalog:

- Run RMAN in the default `NOCATALOG` mode when restoring the datafiles.
- If you must use a recovery catalog because the control file is not large enough to contain all of the backups that you need to restore, then export the catalog and import it into a different schema or database and use the copied recovery catalog for the test restore. Otherwise, the catalog considers the restored database as the current target database.

Table 8–1 describes the impact on the RMAN repository when you are restoring or duplicating to a new host.

Table 8–1 Restoring and Duplicating to a New Host

Command	Catalog?	Effect on Repository
`RESTORE`	yes	If you run `SWITCH` commands after the restore, then RMAN considers the restored database on the new host as the new location of the original target database. If you do not run `SWITCH` commands, then RMAN views the restored datafiles as image copies that are candidates for future restore jobs.

Table 8–1 Restoring and Duplicating to a New Host

Command	Catalog?	Effect on Repository
RESTORE	no	If you run SWITCH commands after the restore, then RMAN considers the restored database as a new target database. This new database, however, has the same DBID as the original. As a result, you cannot later register this duplicate in the same recovery catalog as the original target database. If you do not run SWITCH commands, then the restore operation has no effect on the repository.
DUPLICATE	yes	Generates a new DBID for the duplicate database, which you must manually register in the catalog. After registration, the repository has records of two distinct databases: the target and the duplicate.
DUPLICATE	no	Generates a new DBID for the duplicate database. The repository in the original target control file is unaffected.

Specifying Filenames When Restoring to a New Host

The basic procedure for restoring the database to a new host does not differ substantially from incomplete recovery on the original host. The principal issue is whether the path names of the database files on the new host are going to be the same as the path names of the files on the primary host.

Which restore procedure you should use depends on your situation. If the path names of the restored files will be the same as the original path names, see "Restoring Datafile Copies to a New Host: Example" on page 8-13. If the path names are different, refer to "Performing Disaster Recovery" on page 8-18.

Note the following when restoring to a new host:

- Make the target initialization parameter file accessible on the new host by copying it from the old host to a new host using an operating system utility.
- Make sure backups used for the restore are accessible on the restore host. For example, if the backups were made with a media manager, then make sure the tape device is connected to the new host.
- If the files are in the same location in the new host, then you do not need to recatalog them. If you transfer the files to a new location, then use the CATALOG command to update the RMAN repository with the new filenames and use the CHANGE . . . UNCATALOG command to uncatalog the old filenames.

Determining the SCN for Incomplete Recovery After Restore

Because the restored database will not have the online redo logs of the production database, perform incomplete recovery up to the lowest SCN of the most recently archived log in each thread and then open with the RESETLOGS option. Obtain the SCN for recovery termination by finding the lowest SCN among the most recent archived logs for each thread.

Start SQL*Plus and use the following query to determine the necessary SCN:

```
SQL> SELECT MIN(maxnc) FROM
 (SELECT MAX(a.NEXT_CHANGE#) maxnc
      FROM V$ARCHIVED_LOG a, V$THREAD t
      WHERE a.THREAD# = t.THREAD#
        AND a.ARCHIVED='YES'
        AND t.ENABLED='DISABLED'
      GROUP BY a.THREAD#);
```

Testing the Restore of a Database to a New Host: Scenario

The DUPLICATE command is the preferred method of copying the target database. DUPLICATE creates a new DBID for the copied database, allowing it to be registered in the same recovery catalog as the original target database. However, you may wish to perform a test run of your disaster recovery scenarios that uses exactly the same steps that you would use in a genuine emergency. If so, then you should use the RESTORE and RECOVER commands rather than DUPLICATE.

This scenario assumes the following:

- Two networked machines, hosta and hostb, are running Sun Solaris
- A media management subsystem is accessible by both machines
- The directory structure of hostb is different from hosta, so that trgta is located in /net/hosta/dev3/oracle/dbs, but you want to restore the database to /net/hostb/oracle/oradata/test
- A target database named trgta is on hosta and uses a recovery catalog catdb
- Database trgta uses a server parameter file (not a client-side initialization parameter file)
- You want to test the restore and recovery of trgta on hostb, while keeping database trgta up and running on hosta
- The ORACLE_SID for the trgta database is trgta and will not change for the restored database

- You have recoverable backups on tape of all datafiles
- You have backups of the archived logs required to recover the datafiles
- You have control file and server parameter file autobackups on tape
- You have a record of the DBID for `trgta`

To test the restore of the database to a new host:

1. Make backups of the target database available to `hostb`. To test disaster recovery, you need to have a recoverable backup of the target database. When preparing your disaster recovery strategy, ensure that the backups of the datafiles, control files, and server parameter file are restorable on `hostb`. Hence, you must configure the media management software so that `hostb` is a media manager client and can read the backup sets created on `hosta`. Consult the media management vendor for support on this issue.
2. Configure the `ORACLE_SID` on `hostb`. This case study assumes that you want to authenticate yourself through the operating system, which is much faster than configuring Oracle Net and creating a password file. However, you must be connected to hostb either locally or through telnet.

   While connected to hostb with administrator privileges, edit the `/etc/group` file so that you are included:

   ```
   dba:*:614:<your_user_name>
   ```

   Run the `setenv` command on `hostb` to set the `ORACLE_SID`. In this example, you set the SID to the same value that you used on `hosta`:

   ```
   % setenv ORACLE_SID trgta
   ```

   Start RMAN and connect to the target instance **without** connecting to the recovery catalog.

   ```
   % rman TARGET / NOCATALOG
   ```

3. Start the instance without mounting it. To start the instance, you first need to set the DBID. The DBID is recorded in several places, including:
   - V$DATABASE in the target and RC_DATABASE in the catalog
   - The RMAN output (command-line and V$RMAN_STATUS)
   - The filename of the control file autobackups

Run SET DBID to set the DBID, then run STARTUP NOMOUNT:

```
SET DBID 1340752057;
STARTUP NOMOUNT
```

RMAN will fail to find the server parameter file, which has not yet been restored, but will start the instance with a "dummy" file. Sample output follows:

```
startup failed: ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file
'/net/hostb/oracle/dbs/inittrgta.ora'

trying to start the Oracle instance without parameter files ...
Oracle instance started
```

4. Restore and edit the server parameter file. Because you enabled the control file autobackup feature when making your backups, the server parameter file is included in the backup sets. Hence, you can allocate a channel to the media manager and restore the server parameter file to a new location as a client-side initialization parameter file. Then you can edit the client-side file and restart the instance with the edited client-side file. For example:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS='...';
  RESTORE SPFILE TO PFILE '?/oradata/test/inittrgta.ora' FROM AUTOBACKUP;
  SHUTDOWN ABORT;
}
```

Change any location-specific parameters, for example, those ending in _DEST and _PATH, to reflect the new directory structure. For example, edit the following parameters:

```
- IFILE
- *_DUMP_DEST
- LOG_ARCHIVE_DEST*
- CONTROL_FILES
```

Restart the instance, specifying the client-side initialization parameter file that you restored:

```
STARTUP FORCE NOMOUNT PFILE='?/oradata/test/inittrgta.ora';
```

5. Restore the control file from an autobackup and then mount the database. Because you edited the init.ora in the preceding step, RMAN restores the

control file to whatever location you specified in the CONTROL_FILES initialization parameter. For example:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS='...';
  RESTORE CONTROLFILE FROM AUTOBACKUP;
  ALTER DATABASE MOUNT;
}
```

6. Query the database filenames recorded in the control file on the new host (hostb). Because the control file is from the trgta database, the recorded filenames use the original hosta filenames. You can query V$ views to obtain this information. Start a new SQL*Plus session and connect to the newly created instance on hostb:

   ```
   % sqlplus '/ AS SYSDBA'
   ```

   Run the following query in SQL*Plus:

   ```
   SQL> COLUMN NAME FORMAT a60
   SQL> SPOOL LOG 'db_filenames.out'
   SQL> SELECT FILE# AS "File/Grp#", NAME FROM V$DATAFILE
        UNION
        SELECT GROUP#,MEMBER FROM V$LOGFILE;
   SQL> SPOOL OFF
   SQL EXIT
   ```

7. Restore and recover the database. At this point you are ready to write the RMAN recovery script. The script should do the following:

   – Run SET NEWNAME for each datafile so it is renamed to its new hostb path name

   – Run SQL commands to rename the online redo logs to their new hostb path names

   – Perform a SET UNTIL to limit media recovery to the end of the archived redo logs, as described in "Determining the SCN for Incomplete Recovery After Restore" on page 8-13

   – Run SWITCH so that the control file recognizes the new path names as the official new names of the datafiles

   – Restore and recover the database

The following is an example of an RMAN script to perform these steps, which is contained in text file `reco_test.rman`:

```
RUN
{
  # allocate a channel to the tape device
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS='...';

  # rename the datafiles and online redo logs
  SET NEWNAME FOR DATAFILE 1 TO '?/oradata/test/system01.dbf';
  SET NEWNAME FOR DATAFILE 2 TO '?/oradata/test/undotbs01.dbf';
  SET NEWNAME FOR DATAFILE 3 TO '?/oradata/test/cwmlite01.dbf';
  SET NEWNAME FOR DATAFILE 4 TO '?/oradata/test/drsys01.dbf';
  SET NEWNAME FOR DATAFILE 5 TO '?/oradata/test/example01.dbf';
  SET NEWNAME FOR DATAFILE 6 TO '?/oradata/test/indx01.dbf';
  SET NEWNAME FOR DATAFILE 7 TO '?/oradata/test/tools01.dbf';
  SET NEWNAME FOR DATAFILE 8 TO '?/oradata/test/users01.dbf';
  SQL "ALTER DATABASE RENAME FILE ''/dev3/oracle/dbs/redo01.log''
      TO ''?/oradata/test/redo01.log'' ";
  SQL "ALTER DATABASE RENAME FILE ''/dev3/oracle/dbs/redo02.log''
      TO ''?/oradata/test/redo02.log'' ";

  # Do a SET UNTIL to prevent recovery of the online logs
  SET UNTIL SCN 123456;
  # restore the database and switch the datafile names
  RESTORE DATABASE;
  SWITCH DATAFILE ALL;

  # recover the database
  RECOVER DATABASE;
}
EXIT
```

Caution: It is imperative that you **not** be connected with the recovery catalog when you run this script, so that you do not incorporate extraneous repository data about backups into the recovery catalog.

For example, connect and execute as follows:

```
% rman TARGET / NOCATALOG
RMAN> @reco_test.rman
```

RMAN will apply as many of the archived redo logs as it can and leave the database in a state in which is can be opened.

8. Open the database. From the RMAN prompt, open the database with the RESETLOGS options:

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

9. Remove the test files from the operating system. If the test is successful, then shut down the instance and exit the RMAN session:

```
RMAN> SHUTDOWN ABORT
RMAN> EXIT
```

Remove all test files. You can do this with an operating system utility or in RMAN. For example, in Unix you could perform the procedure this way:

```
% rm $ORACLE_HOME/oradata/test/*
```

You can also use RMAN for a procedure that works ok all platforms. For example:

```
RMAN> STARTUP FORCE NOMOUNT PFILE='?/oradata/test/inittrgta.ora';
RMAN> DROP DATABASE;
```

Because you did not perform the restore and recovery when connected to the recovery catalog, the recovery catalog contains no records for any of the restored files or the procedures performed during the test. Likewise, the control file of the trgta database is completely unaffected by the test.

Performing Disaster Recovery

If you are in a disaster recovery scenario, then presumably you have lost the target database, the recovery catalog database, all control files, all online redo logs, and all parameter files.

To perform a disaster recovery, the minimum required set of backups is backups of some datafiles, some archived redo logs generated after the time of the backup, and at least one autobackup of the control file.

> **See Also:** "Control File and Server Parameter File Autobackups" on page 2-38

The basic procedure for disaster recovery is found in "Performing Recovery with a Backup Control File" on page 8-6, with an additional first step of restoring an

autobackup of the server parameter file as described in *Oracle Database Backup and Recovery Basics*. After the instance is started, you can restore an autobackup of the control file, mount it, then restore and recover the datafiles. Because you are restoring to a new host, you should review the considerations described in "Restoring the Database to a New Host" on page 8-11.

The following scenario restores and recovers the database to the most recently available archived log, which in this example is log 1124 in thread 1. It assumes that:

- You are restoring the database to a new host with the same directory structure.
- You have one tape drive containing backups of all the datafiles and archived redo logs through log 1124, as well as autobackups of the control file and server parameter file.
- You do not use a recovery catalog.

In this scenario, perform the following steps:

1. If possible, restore all relevant network files such as `tnsnames.ora` and `listener.ora` by means of operating system utilities.
2. Start RMAN and connect to the target database. If you do not have the Oracle Net files, then connect through operating system authentication.
3. Specify the DBID for the target database with the `SET DBID` command, as described in "Performing Recovery with a Backup Control File and No Recovery Catalog" on page 8-8.
4. Run the `STARTUP NOMOUNT` command. RMAN attempts to start the instance with a dummy server parameter file.
5. Allocate a channel to the media manager and then run the `RESTORE SPFILE FROM AUTOBACKUP` command.
6. Run `STARTUP FORCE NOMOUNT` mode so that the instance is restarted with the restored server parameter file.
7. Allocate a channel to the media manager and then restore a control file autobackup (refer to"Performing Recovery with a Backup Control File and No Recovery Catalog" on page 8-8).
8. Mount the restored control file.
9. Catalog any backups not recorded in the repository with the `CATALOG` command (refer to"Removing Recovery Catalog Records with Status DELETED" on page 13-11).

10. Restore the datafiles to their original locations. If volume names have changed, then run SET NEWNAME commands before the restore and perform a switch after the restore to update the control file with the new locations for the datafiles (refer to"Performing Disaster Recovery" on page 8-18).
11. Recover the datafiles. RMAN stops recovery when it reaches the log sequence number specified.
12. Open the database in RESETLOGS mode. Only complete this last step if you are certain that no other archived logs can be applied.

```
# Start RMAN and connect to the target database
% rman TARGET SYS/oracle@trgt

# Set the DBID for the target database
RMAN> SET DBID 676549873;
RMAN> STARTUP FORCE NOMOUNT;  # rman starts instance with dummy parameter file
RUN
{
  ALLOCATE CHANNEL t1 DEVICE TYPE sbt;
  RESTORE SPFILE FROM AUTOBACKUP;
}
# Restart instance with restored server parameter file
RMAN> STARTUP FORCE NOMOUNT;

RMAN> RUN
{
  # Manually allocate a channel to the media manager
  ALLOCATE CHANNEL t1 DEVICE TYPE sbt;
  # Restore autobackup of the control file. This example assumes that you have
  # accepted the default format for the autobackup name.
  RESTORE CONTROLFILE FROM AUTOBACKUP;
  #  The set until command is used in case the database
  #  structure has changed in the most recent backups, and you wish to
  #  recover to that point-in-time. In this way RMAN restores the database
  #  to the same structure that the database had at the specified time.
  ALTER DATABASE MOUNT;
  SET UNTIL SEQUENCE 1124 THREAD 1;
  RESTORE DATABASE;
  RECOVER DATABASE;
}
RMAN> ALTER DATABASE OPEN RESETLOGS; # Reset the online logs after recovery
completes
```

The following example of the RUN command shows the same scenario except with new filenames for the restored datafiles:

```
RMAN> RUN
{
  #  If you need to restore the files to new locations, tell Recovery Manager
  #  to do this using SET NEWNAME commands:
  SET NEWNAME FOR DATAFILE 1 TO '/dev/vgd_1_0/rlvt5_500M_1';
  SET NEWNAME FOR DATAFILE 2 TO '/dev/vgd_1_0/rlvt5_500M_2';
  SET NEWNAME FOR DATAFILE 3 TO '/dev/vgd_1_0/rlvt5_500M_3';
  ALLOCATE CHANNEL t1 DEVICE TYPE sbt;
  RESTORE CONTROLFILE FROM AUTOBACKUP;
  ALTER DATABASE MOUNT;
  SET UNTIL SEQUENCE 124 THREAD 1;
  RESTORE DATABASE;
  SWITCH DATAFILE ALL; # Update control file with new location of datafiles.
  RECOVER DATABASE;
}
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Performing Block Media Recovery with RMAN

The BLOCKRECOVER command can restore and recover individual datablocks within a datafile. This procedure is useful when a trace file or standard output reveals that a small number of blocks within a datafile are corrupt.

Block media recovery is not useful in cases where the extent of data loss or corruption is not known; in this case, use datafile recovery instead.

See Also:

- "Block Media Recovery with RMAN" on page 3-10 for an overview of block media recovery,
- *Oracle Database Recovery Manager Reference* for BLOCKRECOVER syntax
- *Oracle Database Reference* for details about the $DATABASE_BLOCK_CORRUPTION view

Recovering Datablocks By Using All Available Backups

In this scenario, you identify the blocks that require recovery and then use any available backup to perform the restore and recovery of these blocks.

To recover datablocks by using all available backups:

1. Obtain the datafile numbers and block numbers for the corrupted blocks. Typically, you obtain this output from the standard output, the `alert.log`, trace files, or a media management interface. For example, you may see the following in a trace file:

```
ORA-01578: ORACLE data block corrupted (file # 8, block # 13)
ORA-01110: data file 8: '/oracle/oradata/trgt/users01.dbf'
ORA-01578: ORACLE data block corrupted (file # 2, block # 19)
ORA-01110: data file 2: '/oracle/oradata/trgt/undotbs01.dbf'
```

2. Assuming that you have preallocated automatic channels, run the BLOCKRECOVER command at the RMAN prompt, specifying the file and block numbers for the corrupted blocks as in the following example:

```
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19;
```

Recovering Datablocks By Using Selected Backups

In this scenario, you identify the blocks that require recovery, and then use only selected backups to perform the restore and recovery of these blocks.

To recover datablocks while limiting the type of backup:

1. Obtain the datafile numbers and block numbers for the corrupted blocks. Typically, you obtain this output from the standard output, the `alert.log`, trace files, or a media management interface. For example, you may see the following in a trace file:

```
ORA-01578: ORACLE data block corrupted (file # 8, block # 13)
ORA-01110: data file 8: '/oracle/oradata/trgt/users01.dbf'
ORA-01578: ORACLE data block corrupted (file # 2, block # 19)
ORA-01110: data file 2: '/oracle/oradata/trgt/undotbs01.dbf'
```

2. Assuming that you have preallocated automatic channels, execute the BLOCKRECOVER command at the RMAN prompt, specifying the file and block numbers for the corrupted blocks and limiting the backup candidates by means of the available options. For example, you can specify what type of backup should be used to restore the blocks:

```
# restore from backupset
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19 FROM BACKUPSET;
# restore from datafile image copy
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19
      FROM DATAFILECOPY;
```

You can indicate the backup by specifying a tag:

```
# restore from backupset with tag "mondayam"
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 199
      FROM TAG = mondayam;
```

You can limit the backup candidates to those made before a certain point:

```
# restore using backups made before one week ago
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19
  RESTORE UNTIL 'SYSDATE-7';
# restore using backups made before SCN 100
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19
  RESTORE UNTIL SCN 100;
# restore using backups made before log sequence 7024
RMAN> BLOCKRECOVER DATAFILE 8 BLOCK 13 DATAFILE 2 BLOCK 19
  RESTORE UNTIL SEQUENCE 7024;
```

Note that if you limit the restore of datablocks with the UNTIL clause, then RMAN must perform more recovery on the blocks, and the recovery phase must scan all logs for changes to the specified blocks.

Recovering Blocks Listed in V$DATABASE_BLOCK_CORRUPTION

The V$DATABASE_BLOCK_CORRUPTION view indicates which blocks in a datafile were marked corrupt since the most recent BACKUP or BACKUP VALIDATE command was run. After a corrupt block is repaired, the row identifying this block is deleted from the view.

You can check for logical corruption in the database by running the BACKUP (with or without VALIDATE option) with the CHECK LOGICAL command. If RMAN finds corrupt blocks, then it populates V$DATABASE_BLOCK_CORRUPTION. The backup will stop if the number of corrupt blocks exceeds MAXCORRUPT. A historical record of block corruptions in RMAN backups is kept in V$BACKUP_CORRUPTION and V$COPY_CORRUPTION.

In this scenario, you identify the blocks that require recovery by querying V$DATABASE_BLOCK_CORRUPTION, and then instruct RMAN to recover all blocks listed in this view by means of the CORRUPTION LIST keyword.

To recover datablocks while limiting the type of backup:

1. Query V$DATABASE_BLOCK_CORRUPTION to determine whether corrupt blocks exist in the most recent backups of the datafiles:

```
SQL> SELECT * FROM V$DATABASE_BLOCK_CORRUPTION;
```

2. Assuming that you have preallocated automatic channels, recover all blocks marked corrupt in V$DATABASE_BLOCK_CORRUPTION by running the BLOCKRECOVER CORRUPTION LIST command. For example, this command restores blocks from backups created more than 10 days ago:

```
BLOCKRECOVER CORRUPTION LIST
  RESTORE UNTIL TIME 'SYSDATE-10';
```

See *Oracle Database Recovery Manager Reference* for more details on block media recovery in RMAN.

RMAN Restore and Recovery Examples

The following sections illustrate the use of RMAN restore and recovery techniques in advanced scenarios.

Restoring Datafile Copies to a New Host: Example

To move the database to a new host by means of datafile copies, you must transfer the copies manually to the new machine. This example assumes that you are using a recovery catalog.

1. After connecting to the target database and recovery catalog, run a LIST command to see a listing of datafile copies and their associated primary keys, as in the following example:

```
LIST COPY;
```

2. Copy the datafile copies to the new host with an operating system utility. For example, in UNIX:

```
% cp -r /tmp/*dbf /net/new_host/oracle/oradata/trgt
```

3. Start RMAN and then uncatalog the datafile copies on the old host. For example, enter:

```
CHANGE COPY OF DATAFILE 1,2,3,4,5,6,7,8 UNCATALOG;
```

4. Catalog the datafile copies, using their new filenames or CATALOG START WITH (if you know all the files are in directories with a common prefix easily addressed with a CATALOG START WITH). For example, run:

```
CATALOG START WITH '?/oradata/trgt/';
```

Or this example specifies files individually:

```
CATALOG DATAFILECOPY
  '?/oradata/trgt/system01.dbf', '?/oradata/trgt/undotbs01.dbf',
  '?/oradata/trgt/cwmlite01.dbf', '?/oradata/trgt/drsys01.dbf',
  '?/oradata/trgt/example01.dbf', '?/oradata/trgt/indx01.dbf',
  '?/oradata/trgt/tools01.dbf', '?/oradata/trgt/users01.dbf';
```

5. Perform the restore and recovery operation described in "Performing Disaster Recovery" on page 8-18.

Restoring When Multiple Databases in the Catalog Share the Same Name: Example

As explained in the description for SET DBID in *Oracle Database Recovery Manager Reference*, you must run the SET DBID command to restore the control file when the target database is not mounted and multiple databases registered in the recovery catalog share the same name. In this case, do the following steps in order:

1. Start RMAN and connect to the target database.
2. Run the STARTUP FORCE NOMOUNT command.
3. Run the SET DBID command to distinguish this connected target database from other target databases that have the same name.
4. Run the RESTORE CONTROLFILE command. After restoring the control file, you can mount the database to restore the rest of the database.

This procedure avoids the RMAN-20005 message when you attempt to restore the control file. This message occurs because more than one target database has the same name, so RMAN requires the unique DBID to distinguishes the databases from one another.

Obtaining the DBID of a Database That You Need to Restore

If you have saved the RMAN output log files, then refer to these logs to determine the database identifier. RMAN automatically provides the DBID whenever you connect to the database:

```
% rman TARGET /

Recovery Manager: Release 10.1.0.2.0 - Production

connected to target database: RMAN (DBID=1231209694)
```

The output from RMAN jobs is also stored persistently in V$RMAN_STATUS and RC_RMAN_STATUS. The DBID is also stored in the RC_DATABASE and RC_DATABASE_INCARNATION recovery catalog views.

Because the names of the databases that are registered in the recovery catalog are presumed nonunique in this scenario, you must use some other unique piece of information to determine the correct DBID. If you know the filename of a datafile or online redo log associated with the database you wish to restore, and this filename is unique across all databases registered in the recovery catalog, then substitute this fully specified filename for *filename_of_log_or_df* in the following queries. Determine the DBID by performing one of the following queries:

```
SELECT DISTINCT DB_ID
FROM DB, DBINC, DFATT
WHERE DB.DB_KEY = DBINC.DB_KEY
  AND DBINC.DBINC_KEY = DFATT.DBINC_KEY
  AND DFATT.FNAME = 'filename_of_log_or_df';

SELECT DISTINCT DB_ID
FROM DB, DBINC, ORL
WHERE DB.DB_KEY = DBINC.DB_KEY
  AND DBINC.DBINC_KEY = ORL.DBINC_KEY
  AND ORL.FNAME = 'filename_of_log_or_df';
```

Restoring a Backup Control File By Using the DBID

To set the DBID, connect RMAN to the target database and run the following SET command, where *target_dbid* is the value you obtained from the previous step:

```
SET DBID = target_dbid;
```

To restore the control file to its default location and then mount it, run:

```
RESTORE CONTROLFILE;
ALTER DATABASE MOUNT;
```

To restore and recover the database, run:

```
RESTORE DATABASE;
RECOVER DATABASE
  # optionally, delete logs restored for recovery and limit disk space used
  DELETE ARCHIVELOG MAXSIZE 2M;
```

Recovering a Database in NOARCHIVELOG Mode: Example

You can recover a database running in NOARCHIVELOG mode with incremental backups. Note that the incremental backups must be consistent, like all backups of a database run in NOARCHIVELOG mode, so you cannot make backups of the database when it is open.

Assume the following scenario:

- You run database trgt in NOARCHIVELOG mode.
- You use a recovery catalog.
- You shut down the database consistently and make a level 0 backup of database trgt to tape on Sunday afternoon.
- You shut down the database consistently and make a level 1 differential incremental backup to tape at 3:00 a.m. on Wednesday and Friday.
- The database has a media failure on Saturday, destroying half of the datafiles as well as the online redo logs.

In this case, you must perform an incomplete media recovery until Friday, since that is the date of the most recent incremental backup. RMAN uses the level 0 Sunday backup as well as the Wednesday and Friday level 1 backups.

Because the online redo logs are lost, you must specify the NOREDO option in the RECOVER command. You must also specify NOREDO if the online logs are available but the redo cannot be applied to the incrementals. If you do not specify NOREDO, then RMAN searches for redo logs after applying the Friday incremental backup, and issues an error message when it does not find them. If the correct online logs for the restored backup had been available, then you could have run RECOVER DATABASE without specifying NOREDO. The changes in the online logs would have been applied.

After connecting to trgt and the catalog database, recover the database with the following command:

```
STARTUP FORCE MOUNT;
RESTORE CONTROLFILE;  # restore control file from consistent backup
ALTER DATABASE MOUNT;
RESTORE DATABASE;  # restore datafiles from consistent backup
RECOVER DATABASE NOREDO;  # specify NOREDO because online redo logs are lost
ALTER DATABASE OPEN RESETLOGS;
```

The recovered database reflects only changes up through the time of the Friday incremental backup. Because there are no archived redo logs, there is no way to recover changes made after the incremental backup.

Recovering a Lost Datafile Without a Backup: Example

RMAN can handle lost datafiles without user intervention during restore and recovery. When a datafile is lost, the possible cases can be classified as follows:

- The control file knows about the datafile, that is, the user backed up the control file after datafile creation, but the datafile itself is not backed up. If the datafile record is in the control file, then `RESTORE` creates the datafile in the original location or in a user-specified location (for example, with `SET NEWNAME`). The `RECOVER` command can then apply the necessary logs to the datafile.
- The control file does not have the datafile record, that is, the user did not back up the control file after datafile creation. During recovery, the database will detect the missing datafile and report it to RMAN, which will create a new datafile and continue recovery by applying the remaining logs. If the datafile was created in a parent incarnation, it will be created during restore or recover as appropriate.

In this example, the following sequence of events occurs:

1. You make a whole database backup of your `ARCHIVELOG` mode database.
2. You create a tablespace `history` containing a single datafile called `/mydb/history01.dbf`.
3. You populate the newly created datafile with data.
4. You archive all the active online redo logs.
5. A user accidentally deletes the datafile `history01.dbf` from the operating system before you have a chance to back it up.

In this case, the current control file knows about the datafile. To restore and recover the datafile, start RMAN, connect to the target database, and then enter the following commands at the RMAN prompt:

```
# take the tablespace with the missing datafile offline
SQL "ALTER TABLESPACE history OFFLINE IMMEDIATE";
# restore the tablespace even though you have no backup
RESTORE TABLESPACE history;
# recover tablespace
RECOVER TABLESPACE hisotry;
# bring the recovered tablespace back online
```

```
SQL "ALTER TABLESPACE history ONLINE";
```

Transporting a Tablespace to a Different Database on the Same Platform: Example

You can use the transportable tablespace feature to copy a tablespace from one database to another database. As described in *Oracle Database Administrator's Guide*, the basic method for transporting tablespaces to a database on the same platform does not make use of RMAN. Nevertheless, if you use RMAN to back up your target database, then you can also use RMAN to transport backups of a tablespace from one database into another, following the procedure described in this section.

Reasons for using this procedure instead of ordinary tablespace transport include:

- The tablespace cannot be made read-only currently on the target database
- The current contents of the tablespace are not the desired ones, but there is a backup which could be restored and recovered to a point in time where the data is the version needed for transport
- The target host is too busy and cannot sustain the overhead involved with copying the tablespace
- There is no space to make a copy of the tablespace to be transported on the target host.

In the following procedure, assume that:

- You are transporting a backup of tablespace `users` from database `trgta`, located on computer `hosta`, to database `trgtb`, located on computer `hostb`
- Both hosts run the same operating system.
- The platform being used is Solaris or another similar Unix system. (For other platforms, file naming conventions and steps performed at the host operating system level may need different syntax.)
- You want to name the restored backup of the `users` datafile as `/net/hostb/oracle/oradata/trgtb/users01.dbf` in database `trgtb`
- You have recoverable backups of the following files from database `trgta`:
  - All datafiles in the tablespaces that you are transporting (in this example, the `users` tablespace)
  - Datafiles in the `SYSTEM` tablespace
  - Datafiles with rollback or undo segments
  - A control file that contains the metadata for the preceding datafile backups

- The backups of `trgta` are accessible by `hostb` through a tape device.

  See Also: *Oracle Database Administrator's Guide* to learn how to transport a tablespace

To transport a tablespace into a different database:

1. Create an auxiliary instance on `hostb` according to the instructions in the "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9.
2. Connect RMAN to the auxiliary instance as if it were a new target instance. For example:

   ```
   rman TARGET SYS/oracle@auxdb CATALOG rman/rman@catdb
   ```

3. Restore the control file to a temporary location, then mount the control file and exit the session. For example:

   ```
   RESTORE CONTROLFILE TO '/net/hostb/tmp/cf.f';
   STARTUP FORCE MOUNT;
   EXIT
   ```

4. Reconnect RMAN to the same auxiliary instance in `NOCATALOG` mode. You connect in `NOCATALOG` so that you do not pollute the recovery catalog with unecessary metadata about the restored files. For example:

   ```
   rman TARGET SYS/oracle@auxdb NOCATALOG
   ```

5. Restore and recover the auxiliary database. Perform the following steps:

   a. Specify the past point in time, SCN, or archived log sequence number to which you want to recover the tablespace. You cannot recover the tablespace to the current time. Use the specified `UNTIL` time to indicate which backup of the tablespace that you want to restore.

   b. If the restored control file does not included configured channels, then manually allocate a channel to the device containing the backups.

   c. Run `SET NEWNAME` to specify temporary filenames for the `SYSTEM` datafiles and the datafiles containing rollback or undo segments.

   d. Run `SET NEWNAME` to specify the filenames in the `trgtb` database that will be used by the datafiles in the transported tablespace.

   e. Restore and recover the tablespaces.

   For example, run the following commands:

```
RUN
{
  SET UNTIL ARCHIVELOG 1243 THREAD 1;  # set the end recovery log
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt; # allocate channels if not configured
  # specify temporary name for SYSTEM datafile
  SET NEWNAME FOR DATAFILE 1 TO '/net/hostb/tmp/df1.dbf';
  # specify temporary names for datafiles with undo or rollback segments
  SET NEWNAME FOR DATAFILE 2 TO '/net/hostb/tmp/df2.dbf';
  # specify names for datafiles to be plugged into trgtb database
  SET NEWNAME FOR DATAFILE 8 TO
    '/net/hostb/oracle/oradata/trgtb/users01.dbf';
  # restore and recover the datafiles
  RESTORE DATAFILE 1, 2, 8;
  SWITCH DATAFILE ALL;   # points control file to SET NEWNAME filenames
  RECOVER DATAFILE 1, 2, 8;
}
```

6. Take all auxiliary tablespaces offline except the tablespaces that you recovered in the preceding step. For example:

   ```
   SQL 'ALTER TABLESPACE cwmlite,drsys,example,indx,tools OFFLINE IMMEDIATE';
   ```

7. Open the auxiliary database with the RESETLOGS option. For example:

   ```
   ALTER DATABASE OPEN RESETLOGS;
   ```

8. Make the tablespace that you are transporting into `trgtb` read-only. For example:

   ```
   SQL 'ALTER TABLESPACE users READ ONLY';
   ```

9. Export the metadata from the transported `users` tablespace as described in "Step 2: Generate a Transportable Tablespace Set" in *Oracle Database Administrator's Guide*. For example:

   ```
   exp TRANSPORT_TABLESPACE=y TABLESPACES=(users)
           TRIGGERS=y CONSTRAINTS=n GRANTS=n FILE=expdat.dmp
   ```

10. Shut down the auxiliary instance, and then delete all auxiliary files except the datafiles in the transported tablespace. For example:

    ```
    % sqlplus SYS/oracle@auxdb
    SQL> SHUTDOWN ABORT
    SQL> EXIT
    % rm /net/hostb/tmp/*
    ```

11. Import the metadata from the transported tablespace into the `trgtb` database as described in "Step 4: Plug In the Tablespace Set" in *Oracle Database Administrator's Guide*. For example:

```
imp TRANSPORT_TABLESPACE=y FILE=expdat.dmp
  DATAFILES=('/net/hostb/oracle/oradata/trgtb/users01.dbf')
  TABLESPACES=(users) TTS_OWNERS=(usera)
  FROMUSER=(usera) TOUSER=(userb)
```

Note: You cannot make a backup of a transported tablespace until after it has been opened read / write.

9

Flashback Technology: Recovering from Logical Corruptions

This chapter describes how to use the flashback features of Oracle to retrieve lost data in data recovery scenarios. This chapter includes the following sections:

Oracle Flashback Technology: Overview

Oracle Flashback Technology provides a set of features that support viewing and rewinding data back and forth in time. The flashback features offer the capability to query past versions of schema objects, query historical data, analyze database changes, or perform self-service repair to recover from logical corruptions while the database is online.

- **Oracle Flashback Query** feature lets you specify a target time and then run queries against your database, viewing results as they would have appeared at that time. To recover from an unwanted change like an erroneous update to a table, a user could choose a target time before the error and run a query to retrieve the contents of the lost rows.
- **Oracle Flashback Version Query** lets you view all the versions of all the rows that ever existed in one or more tables in a specified time interval. You can also retrieve metadata about the differing versions of the rows, including start time, end time, operation, and transaction ID of the transaction that created the version. This feature can be used both to recover lost data values and to audit changes to the tables queried.
- **Oracle Flashback Transaction Query** lets you view changes made by a single transaction, or by all the transactions during a period of time.
- **Oracle Flashback Table** returns a table to its state at a previous point in time. You can restore table data while the database is online, undoing changes only to the specified table.
- **Oracle Flashback Drop** reverses the effects of a `DROP TABLE` statement.
- **Oracle Flashback Database** provides a more efficient alternative to database point-in-time recovery. When you use flashback database, your current datafiles revert to their contents at a past time. The result is much like the result of a point-in-time recovery using datafile backups and redo logs, but you do not have to restore datafiles from backup and you do not have to re-apply as many individual changes in the redo logs as you would have to do in conventional media recovery.

Flashback Table, Flashback Query, Flashback Transaction Query and Flashback Version Query all rely on **undo data**, records of the effects of each update to an Oracle database and values overwritten in the update. Used primarily for such purposes as providing read consistency for SQL queries and rolling back transactions, these undo records contain the information required to reconstruct

data as it stood at a past time and examine the record of changes since that past time.

See Also:

- *Oracle Database Concepts* and *Oracle Database Administrator's Guide* for more information on undo data and automatic undo management
- "Oracle Flashback Drop: Undo a DROP TABLE Operation" on page 9-6 for more information on Flashback Drop and the recycle bin
- *Oracle Database Application Developer's Guide - Fundamentals* for more information on Flashback Query, Flashback Transaction Query and Flashback Version Query

Oracle Flashback Query: Recovering at the Row Level

In a data recovery context, it is useful to be able to query the state of a table at a previous time. If, for instance, you discover that at 12:30 PM, an employee 'JOHN' had been deleted from your EMPLOYEE table, and you know that at 9:30AM that employee's data was correctly stored in the database, you could query the contents of the table as of a time before the deletion to find out what data had been lost, and, if appropriate, re-insert the lost data in the database.

Querying the past state of the table is achieved using the AS OF clause of the SELECT statement. For example, the following query retrieves the state of the employee record for 'JOHN' at 9:30AM, April 4, 2003:

```
SELECT * FROM EMPLOYEE AS OF TIMESTAMP
   TO_TIMESTAMP('2003-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
   WHERE name = 'JOHN';
```

Restoring John's information to the table EMPLOYEE requires the following update:

```
INSERT INTO employee
    (SELECT * FROM employee AS OF TIMESTAMP
     TO_TIMESTAMP('2003-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
     WHERE name = 'JOHN');
```

The missing row is re-created with its previous contents, with minimal impact to the running database.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for a more extensive discussion of the use of the SELECT... AS OF SQL statement and extensive examples of its use.
- *Oracle Database SQL Reference* for more details on the syntax of the SELECT... AS OF form of the SELECT statement.

Oracle Flashback Table: Returning Individual Tables to Past States

Oracle Flashback Table provides the DBA the ability to recover a table or set of tables to a specified point in time in the past very quickly, easily, and without taking any part of the database offline. In many cases, Flashback Table eliminates the need to perform more complicated point-in-time recovery operations. Flashback Table restores tables while automatically maintaining associated attributes such as current indexes, triggers and constraints, and not requiring the DBA to find and restore application-specific properties. Using Flashback Table causes the contents of one or more individual tables to revert to their state at some past SCN or time.

Flashback Table uses information in the undo tablespace to restore the table. This provides significant benefits over media recovery in terms of ease of use, availability and faster restoration of data.

For more information on Automatic Undo Management, see *Oracle Database Administrator's Guide.*

Prerequisites for Using Flashback Table

The prerequisites for performing a FLASHBACK TABLE operation are as follows:

- You must have been granted the FLASHBACK ANY TABLE system privilege or you must have the FLASHBACK object privilege on the table.
- You must have SELECT, INSERT, DELETE, and ALTER privileges on the table.
- Undo information retained in the undo tablespace must go far enough back in time to satisfy the specified target point in time or SCN for the FLASHBACK TABLE operation.
- Row movement must be enabled on the table for which you are issuing the FLASHBACK TABLE statement. You can enable row movement with the following SQL statement:

```
ALTER TABLE table ENABLE ROW MOVEMENT;
```

Performing Flashback Table

The following SQL*Plus statement performs a FLASHBACK TABLE operation on the table employee:

```
FLASHBACK TABLE employee TO TIMESTAMP
     TO_TIMESTAMP('2003-04-04 09:30:00', 'YYYY-MM-DD HH24:MI:SS');
```

The employee table is restored to its state when the database was at the time specified by the timestamp.

You can also specify the target point in time for the FLASHBACK TABLE operation using an SCN:

```
FLASHBACK TABLE employee TO SCN 123456;
```

The default for a FLASHBACK TABLE operation is for triggers on a table to be disabled. The database disables triggers for the duration of the operation, and then returns them to the state that they were in before the operation was started. If you wish for the triggers to stay enabled, then use the ENABLE TRIGGERS clause of the FLASHBACK TABLE statement, as shown in this example:

```
FLASHBACK TABLE t1 TO TIMESTAMP '2003-03-03 12:05:00' ENABLE TRIGGERS;
```

The following scenario is typical of the kind of logical corruption where Flashback Table could be used:

At 17:00 an HR administrator discovers that an employee "JOHN" is missing from the EMPLOYEE table. This employee was present at 14:00, the last time she ran a report. Someone accidentally deleted the record for "JOHN" between 14:00 and the present time. She uses Flashback Table to return the table to its state at 14:00, as shown in this example:

```
FLASHBACK TABLE EMPLOYEES TO TIMESTAMP
      TO_TIMESTAMP('2003-04-04 14:00:00','YYYY-MM-DD HH:MI:SS')
      ENABLE TRIGGERS;
```

> **See Also:** *Oracle Database SQL Reference* for a simple Flashback Table scenario

Oracle Flashback Drop: Undo a DROP TABLE Operation

Oracle Flashback Drop reverses the effects of a DROP TABLE operation. The intention behind this feature is to provide users with a recovery mechanism for an accidental drop of a table. Flashback Drop is substantially faster than other recovery mechanisms (such as point-in-time recovery) and also does not lead to any loss of recent transactions.

When you drop a table, the database does not immediately remove the space associated with the table. Instead, the table is renamed and, along with any associated objects, it is placed in the **Recycle Bin** of the database. The Flashback Drop operation recovers the table from the recycle bin.

To understand how to use Oracle Flashback Drop, you must also understand how the recycle bin works, and how to access and manage its contents.

This section covers the following topics:

What is the Recycle Bin?

The recycle bin is a logical container for all dropped tables and their dependent objects. When a table is dropped, the database will store the table, along with its dependent objects in the recycle bin so that they can be recovered later. Dependent objects which are stored in the recycle bin include indexes, constraints, triggers, nested tables, LOB segments and LOB index segments.

How Tables and Other Objects Are Placed in the Recycle Bin

Tables are placed in the recycle bin along with their dependent objects whenever a DROP TABLE statement is executed. For example, this statement places the EMPLOYEE_DEMO table, along with any indexes, constraints, or other dependent objects listed previously, in the recycle bin:

```
SQL> DROP TABLE EMPLOYEE_DEMO;
Table Dropped
```

The table and its dependent objects will remain in the recycle bin until they are **purged** from the recycle bin. You can explicitly purge a table or other object from the recycle bin with the SQL*Plus PURGE statement, as described in "Purging Objects from the Recycle Bin" on page 9-12. If you are sure that you will not want to recover a table later, you can drop it immediately and permanently, instead of placing it in the recycle bin, by using the PURGE option of the DROP TABLE statement, as shown in this example:

```
DROP TABLE employee_demo PURGE;
```

Even if you do not purge objects from the recycle bin, the database purges objects from the recycle bin to meet tablespace space constraints. See "Recycle Bin Capacity and Space Pressure" on page 9-9 for more details.

Recycle bin objects are not counted as used space. If you query the space views to obtain the amount of free space in the database, objects in the recycle bin are counted as free space.

Dropped objects still appear in the views USER_TABLES, ALL_TABLES, DBA_TABLES, USER_INDEX, ALL_INDEX and DBA_INDEX.A new column, DROPPED, is set to YES for these objects. You can use the DROPPED column in queries against these views to view only objects that are not dropped.

To view only objects in the recycle bin, use the USER_RECYCLEBIN and DBA_RECYCLEBIN views, described later in this chapter.

Naming Convention for Objects in the Recycle Bin

When a table and its dependent objects are moved to the recycle bin, they are assigned unique names, to avoid name conflicts that may arise in the following circumstances:

- A user drops a table, creates another with the same name, then drops the second table.
- Two users have tables with the same name, and both users drop their tables.

The assigned names are globally unique and are used to identify the objects while they are in the recycle bin. Object names are formed as follows:

```
BIN$$globalUID$version
```

where:

- *globalUID* is a globally unique, 24 character long identifier generated for the object.
- *version* is a version number assigned by the database

The recycle bin name of an object is always 30 characters long.

Note that the *globalUID* used in the recycle bin name is not readily correlated with any externally visible piece of information about the object or the database.

Viewing and Querying Objects in the Recycle Bin

You can view the contents of the recycle bin using the SQL*Plus command SHOW RECYCLEBIN.

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                   OBJECT TYPE  DROP TIME
---------------- -------------------------------- ------------ ------------------
EMPLOYEE_DEMO    BIN$gk3lsj/3akk5hg3j2lkl5j3d==$0   TABLE        2003-06-11:17:08:54
```

The ORIGINAL NAME column shows the orignal name of the object, while the RECYCLEBIN NAME column shows the name of the object as it exists in the recycle bin. Use the RECYCLEBIN NAME when issuing queries against tables in the recycle bin.

The database also provices two views for obtaining information about objects in the recycle bin:

View	Description
USER_RECYCLEBIN	Lets users see their own dropped objects in the recycle bin. It has a synonym RECYCLEBIN, for ease of use.
DBA_RECYCLEBIN	Lets administrators see all dropped objects in the recycle bin

This example uses the views to determine the original names of dropped objects:

```
SQL> SELECT object_name as recycle_name, original_name, object_type
     FROM recyclebin;

RECYCLE_NAME                     ORIGINAL_NAME          OBJECT_TYPE
-------------------------------  --------------------  -------------
BIN$gk3lsj/3akk5hg3j2lkl5j3d==$0  EMPLOYEE_DEMO          TABLE
BIN$JKS983293M1dsab4gsz/I249==$0  I_EMP_DEMO            INDEX
BIN$NR72JJN38KM1dsaM4gI348as==$0  LOB_EMP_DEMO          LOB
BIN$JKJ399SLKnaslkJSLK330SIK==$0  LOB_I_EMP_DEMO        LOB INDEX
```

You can query objects that are in the recycle bin, just as you can query other objects, if these three conditions are met:

- You must have the FLASHBACK privilege.
- You must have the privileges that were required to perform queries against the object before it was placed in the recycle bin.
- You must use the recycle bin name of the object in your query, rather than the object's original name.

This example shows the required syntax:

```
SQL> SELECT * FROM "BIN$KSD8DB9L345KLA==$0";
```

(Note the use of quotes due to the special characters in the recycle bin name.)

You can also use Oracle Flashback Query on tables in the recycle bin (again, assuming that you have the privileges described previously).

Recycle Bin Capacity and Space Pressure

There is no fixed amount of space pre-allocated for the recycle bin. Therefore, there is no guaranteed minimum amount of time during which a dropped object will remain in the recycle bin.

The rules that govern how long an object is retained in the recycle bin and how and when space is reclaimed are explained in this section.

Understanding Space Pressure

Dropped objects are kept in the recycle bin until such time as no new extents can be allocated in the tablespace to which the objects belong without growing the tablespace. This condition is referred to as **space pressure**. Space pressure can also arise due to user quotas defined for a particular tablespace. A tablespace may have free space, but the user may have exhausted his or her quota on it.

Oracle never automatically reclaims space or overwrites objects in the recycle bin unless forced to do so in response to space pressure.

How the Database Responds to Space Pressure

When space pressure arises, the database selects objects for automatic purging from the recycle bin. Objects are selected for purging on a first-in, first-out basis, that is, the first objects dropped are the first selected for purging.

Actual purging of objects is done only as needed to meet ongoing space pressure, that is, the databases purges the minimum possible number of objects selected for purging to meet immediate needs for space. This policy serves two purposes:

- It minimizes the performance penalty on transactions that encounter space pressure, by not reclaiming more than is required;
- It maximizes the length of time objects remain in the recycle bin, by leaving them there until space is needed.

Dependent objects such as indexes on a table are selected for purging before the associated table (or other required segment).

If space pressure is due to an individual user's quota on a tablespace being exhausted, the recycle bin purges objects belonging to the tablespace which count against that user's space quotas.

For AUTO EXTEND-able tablespaces, objects are purged from the recycle bin to reclaim space before datafiles are extended.

Recycle Bin Objects and Segments

The recycle bin operates at the object level, in terms of tables, indexes, and so on. An object may have multiple segments associated with it, such as partitioned tables, partitioned indexes, lob segments, nested tables, and so on. Because the database reclaims only the segments needed to immediately satisfy space pressure, it can happen that some but not all segments of an object are reclaimed. When this happens, any segments of the object not reclaimed immediately are marked as temporary segments. These temporary segments are the first candidates to be reclaimed the next time space pressure arises.

In such a case, the partially-reclaimed object can no longer be removed from the recycle bin with Flashback Drop. (For example, if one partition of a partitioned table is reclaimed, the table can no longer be the object of a Flashback Drop.)

Performing Flashback Drop on Tables in the Recycle Bin

Use the FLASHBACK TABLE ... TO BEFORE DROP statement to recover objects from the recycle bin. You can specify either the name of the table in the recycle bin or the original table name. This can be obtained from either the DBA_RECYCLEBIN or USER_RECYCLEBIN view as shown in "Viewing and Querying Objects in the Recycle Bin" on page 9-8. To use the FLASHBACK TABLE ... TO BEFORE DROP statement, you need the same privileges you need to drop the table.

The following example restores the BIN$KSD8DB9L345KLA==$0 table, changes its name back to hr.int_admin_emp, and purges its entry from the recycle bin:

```
FLASHBACK TABLE "BIN$KSD8DB9L345KLA==$0" TO BEFORE DROP;
```

You can also use the table's original name in the Flashback Drop operation:

```
FLASHBACK TABLE HR.INT_ADMIN_EMP TO BEFORE DROP;
```

You can assign a new name to the restored table by specifying the RENAME TO clause. For example:

```
FLASHBACK TABLE "BIN$KSD8DB9L345KLA==$0" TO BEFORE DROP
     RENAME TO hr.int2_admin_emp;
```

Flashback Drop of Multiple Objects With the Same Original Name

You can create, and then drop, several objects with the same original name, and they will all be stored in the recycle bin. For example, consider these SQL statements:

```
CREATE TABLE EMP ( ...columns ); # EMP version 1
DROP TABLE EMP;
CREATE TABLE EMP ( ...columns ); # EMP version 2
DROP TABLE EMP;
CREATE TABLE EMP ( ...columns ); # EMP version 3
DROP TABLE EMP;
```

In such a case, each table EMP is assigned a unique name in the recycle bin when it is dropped. You can use a FLASHBACK TABLE... TO BEFORE DROP statement with the original name of the table, as shown in this example:

```
FLASHBACK TABLE EMP TO BEFORE DROP;
```

The most recently dropped table with that original name is retrieved from the recycle bin, with its original name. You can retrieve it and assign it a new name using a RENAME TO clause. The following example shows the retrieval from the recycle bin of all three dropped EMP tables from the previous example, with each assigned a new name:

```
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_3;
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_2;
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_1;
```

Note that the last table dropped is the first one to be retrieved.

You can also retrieve any table you want from the recycle bin, regardless of any such name collisions, by using the table's unique recycle bin name.

Purging Objects from the Recycle Bin

The PURGE command is used to permanently purge objects from the recycle bin. Once purged, objects can no longer be retrieved from the bin using Flashback Drop.

There are a number of forms of the PURGE statement, depending on exactly which objects you want to purge from the recycle bin

> **See Also:** *Oracle Database SQL Reference* for more information on the PURGE statement

PURGE TABLE: Purging a Table and Dependent Objects

The PURGE TABLE command purges an individual table and all of its dependent objects from the recycle bin. This example shows the syntax, using the table's original name:

```
PURGE TABLE EMP;
```

You can also use the recycle bin name of an object with PURGE TABLE:

```
PURGE TABLE "BIN$KSD8DB9L345KLA==$0";
```

If you have created and dropped multiple tables with the same orignal name, then when you use the PURGE TABLE statement the first table dropped will be the one to be purged.

For example, consider the following series of CREATE TABLE and DROP TABLE statements:

```
CREATE TABLE EMP;   # version 1 of the table
DROP TABLE EMP;     # version 1 dropped
CREATE TABLE EMP;   # version 2 of the table
DROP TABLE EMP;     # version 2 dropped
CREATE TABLE EMP;   # version 3 of the table
DROP TABLE EMP;     # version 3 dropped
```

There are now three EMP tables in the recycle bin. If you execute PURGE TABLE EMP several times, the effect is as described here:

```
PURGE TABLE EMP;    # version 1 of the table is purged
PURGE TABLE EMP;    # version 2 of the table is purged
PURGE TABLE EMP;    # version 3 of the table is purged
```

Note that this is the opposite of the behavior of FLASHBACK TABLE... TO BEFORE DROP, where using the original name of the table retrieves the most recently dropped version from the recycle bin.

PURGE INDEX: Freeing Space in the Recycle Bin

You can use PURGE INDEX to purge just an index for a table, while keeping the base table in the recycle bin. The syntax for purging an index is as follows:

```
PURGE INDEX "BIN$GTE72KJ22H9==$0";
```

By purging indexes from the recycle bin, you can reduce the chance of space pressure, so that dropped tables can remain in the recycle bin longer. If you retrieve a table from the recycle bin using Flashback Drop, you can rebuild the indexes after you retrieve the table.

PURGE TABLESPACE: Purging All Objects in a Tablespace

You can use the PURGE TABLESPACE command to purge all dropped tables and other dependent objects from a specific tablespace. The syntax is as follows:

```
PURGE TABLESPACE hr;
```

You can also purge only objects from a tablespace belonging to a specific user, using the following form of the command:

```
PURGE TABLESPACE hr USER scott;
```

PURGE RECYCLEBIN: Purging All Objects in a User's Recycle Bin

The PURGE RECYCLEBIN command purges the contents of the recycle bin for the currently logged-in user.

```
PURGE RECYCLEBIN;
```

It purges all tables and their dependent objects for this user, along with any other indexes owned by this user but not on tables owned by the user.

PURGE DBA_RECYCLEBIN: Purging All Recycle Bin Objects

If you have the SYSDBA privilege, then you can purge all objects from the recycle bin, regardless of which user owns the objects, using this command:

```
PURGE DBA_RECYCLEBIN;
```

Dropping a Tablespace, Cluster, User or Type and the Recycle Bin

When a tablespace is dropped including its contents, the objects in the tablespace are dropped immediately, and not placed in the recycle bin. Any objects in the recycle bin from the dropped tablespace are purged from the recycle bin.

If all objects from a tablespace have been placed in the recycle bin, then dropping the tablespace causes the objects to be purged, even if you do not use the `INCLUDING CONTENTS` clause with `DROP TABLESPACE`.

When a user is dropped, any objects belonging to the user that are not in the recycle bin are dropped immediately, not placed in the recycle bin. Any objects in the recycle bin that belonged to the user are purged from the recycle bin.

When you drop a cluster, all tables in the cluster are purged. When you drop a user-defined data type, all objects directly or indirectly dependent upon that type are purged.

Privileges and Security

This section summariezes the system privileges required for the operations related to Flashback Drop and the recycle bin.

- `DROP`

  Any user with drop privileges over the object can drop the object, placing it in the recycle bin.

- `FLASHBACK TABLE... TO BEFORE DROP`

  Privileges are tied to the privileges for `DROP`. That is, any user who can drop an object can perform Flashback Drop.

- `PURGE`

  Privileges are tied to the DROP privileges. Any user having DROP TABLE or DROP ANY TABLE privileges can purge the objects from the recycle bin.

- `SELECT` for objects in the Recycle Bin

  Users must have `SELECT` and `FLASHBACK` privileges over an object in the recycle bin to be able to query the object in the recycle bin. Any users who had the `SELECT` privilege over an object before it was dropped continue to have the `SELECT` privilege over the object in the recycle bin.Users must have `FLASHBACK` privilege to query any object in the recycle bin, because these are objects from a past state of the database.

Limitations and Restrictions on Flashback Drop

- The recycle bin functionality is only available for non-system, locally managed tablespaces. If a table is in a non-system, locally managed tablespace, but one or more of its dependent segments (objects) is in a dictionary-managed tablespace, then these objects are protected by the recycle bin.
- There is no fixed amount of space allocated to the recycle bin, and no guarantee as to how long dropped objects remain in the recycle bin. Depending upon system activity, a dropped object may remain in the recycle bin for seconds, or for months.
- While Oracle permits queries against objects stored in the recycle bin, you cannot use DML or DDL statements on objects in the recycle bin.
- You can perform Flashback Query on tables in the recycle bin, but you must use the recycle bin name. You cannot use the original name of the table.
- A table and all of its dependent objects (indexes, LOB segments, nested tables, triggers, constraints and so on) go into the recycle bin together, when you drop the table. Likewise, when you perform Flashback Drop, the objects are generally all retrieved together.

  It is possible, however, that some dependent objects such as indexes may have been reclaimed due to space pressure. In such cases, the reclaimed dependent objects are not retrieved from the recycle bin.
- Due to security concerns, tables which have Fine-Grained Auditing (FGA) and Virtual Private Database (VPD) policies defined over them are not protected by the recycle bin.
- Partitioned index-organized tables are not protected by the recycle bin.
- The recycle bin does not preserve referential constraints on a table (though other constraints will be preserved if possible). If a table had referential constraints before it was dropped (that is, placed in the recycle bin), then you may re-create any referential constraints after you perform Flashback Drop to retrieve the table from the recycle bin.

Oracle Flashback Database: Alternative to Point-In-Time Recovery

Oracle Flashback Database, accessible from both RMAN (by means of the FLASHBACK DATABASE command) and SQL*Plus (by means of the FLASHBACK DATABASE statement), lets you quickly recover the entire database from logical data corruptions or user errors.

It is similar to conventional point in time recovery in its effects, allowing you to return a database to its state at a time in the recent past. Flashback Database is, however, much faster than point-in-time recovery, because it does not require restoring datafiles from backup and it requires applying fewer changes from the archived redo logs.

To enable Flashback Database, you set up a flash recovery area, and set a **flashback retention target**, to specify how far back into the past you want to be able to restore your database with Flashback Database.

From that time on, at regular intervals, the database copies images of each altered block in every datafile into **flashback logs** stored in the flash recovery area. These block images can later be re-used to reconstruct the datafile contents as of any moment at which logs were captured.

To restore a database to its state at some past target time using Flashback Database, each block is restored to its contents as of the flashback logging time most immediately prior to the desired target time, and then changes from the redo logs are applied to fill in changes between the time captured by the flashback logs and the target time. Redo logs must be available for the entire time period spanned by the flashback logs, whether on tape or on disk. In practice, however, redo logs are often kept much longer than flashback logs, so this requirement is not a real limitation.

The time required to perform Flashback Database is largely a function of how far back the target time is and the number of blocks changed, rather than the volume of individual updates to the database.

Limitations of Flashback Database

Because Flashback Database works by undoing changes to the datafiles that exist at the moment that you run the command, it has the following limitations:

- Flashback Database cannot by itself recover from media failure, that is, the corruption or deletion of a datafile. The datafile to be reverted to a past state with flashback database must be present and must not be corrupted due to media failure. However, you can combine Flashback Database and point-in-time recovery in some situations for more efficient recovery. Tablespaces affected by the deletion of a datafile must be restored from backup and recovered to a point in time before the deletion. Tablespaces not affected by the deletion, however, can be reverted to that same point in time using Flashback Database.

- You cannot use Flashback Database if the database's control file has been restored from backup or re-created after Flashback Database was turned on.
- You cannot use Flashback Database to undo a shrink datafile operation.
- You cannot use Flashback Database to return the database to an SCN prior to the earliest SCN available in the flashback logs that currently exist in the flash recovery area. Because flashback logs can be deleted from the flash recovery area when space is needed, it is important to size your flash recovery area appropriately.

Finally, it is important to note that the flashback retention target is a target, not an absolute guarantee that Flashback Database will be available. If your flash recovery area is not large enough to hold both required files such as archived redo logs and other backups, flashback logs may be deleted to make room in the flash recovery area for these required files. If you discover that Oracle has discarded flashback logs required to reach your desired target time for Flashback Database, you can always use traditional point-in-time recovery instead to achieve a similar result.

Requirements for Flashback Database

The requirements for enabling Flashback Database are:

- Your database must be running in `ARCHIVELOG` mode, because archived logs are used in the Flashback Database operation.
- You must have a flash recovery area enabled, because flashback logs can only be stored in the flash recovery area.
- For Real Application Clusters databases, the flash recovery area must be stored in a clustered file system or in ASM.

Enabling Flashback Database

To enable Flashback Database, set the `DB_FLASHBACK_RETENTION_TARGET` initialization parameter and issue the `ALTER DATABASE FLASHBACK ON` statement. Follow the process outlined here.

1. Start SQL*Plus and ensure that the database is mounted, but not open. For example:

   ```
   SQL> SELECT STATUS FROM V$INSTANCE;
   ```

2. By default the flashback retention target is set to one day (1440 minutes). If you wish, you can change the retention target. For example, if you want to retain

enough flashback logs to be able to perform a 72 hour flashback, set the retention target to 4320 minutes (3 days x 24 hours / day x 60 minutes / hour):

```
SQL> ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET=4320;
```

3. Enable the Flashback Database feature for the whole database:

```
SQL> ALTER DATABASE FLASHBACK ON;
```

By default, flashback logs are generated for all permanent tablespaces. If you wish, you can reduce overhead by disabling flashback logging specific tablespaces:

```
SQL> ALTER TABLESPACE test1 FLASHBACK OFF;
```

You can re-enable flashback logging for a tablespace later with this command:

```
SQL> ALTER TABLESPACE test1 FLASHBACK ON;
```

Note that if you disable Flashback Database for a tablespace, then you must take its datafiles offline before running FLASHBACK DATABASE.

You can disable flashback logging for the entire database with this command:

```
SQL> ALTER DATABASE FLASHBACK OFF;
```

You can enable Flashback Database not only on a primary database, but also on a standby database. Enabling Flashback Database on a standby database allows one to perform Flashback Database on the standby database. Flashback Database of standby databases has a number of applications in the Data Guard environment. See *Oracle Data Guard Concepts and Administration* for details.

Sizing the Flash Recovery Area for Flashback Logs

The setting of the DB_FLASHBACK_RETENTION_TARGET initialization parameter determines, indirectly, how much flashback log data the database should keep. This limit is contingent upon sufficient space existing in the flash recovery area. The size of flashback logs can vary considerably, however, depending on the locality of database changes during a given flashback logging interval.

Estimating Flashback Database Storage Requirements

The V$FLASHBACK_DATABASE_LOG view can help you decide how much space to add to your flash recovery area for flashback logs. After you have enabled the Flashback Database feature and allowed the database to generate some flashback logs, run the following query:

```
SQL> SELECT ESTIMATED_FLASHBACK_SIZE FROM V$FLASHBACK_DATABASE_LOG;
```

An estimate of disk space needed to meet the current flashback retention target is calculated, based on the database workload since Flashback Database was enabled. Add the amount of disk space specified in $FLASHBACK_DATABASE_LOG.ESTIMATED_FLASHBACK_SIZE to your flash recovery area size, to hold the dataabase flashback logs.

Space usage in the flash recovery area is always balanced among backups and archived logs which must be kept according to the retention policy, and other files like flashback logs and backups already moved to tape but still cached on disk. If you have not allocated enough space in your flash recovery area to store your flashback logs and still meet your other backup retention requirements, flashback logs may be deleted from the recovery area to make room for other required files. In such situations you will still be able to use point-in-time recovery to revert your database to a previous state.

Determining the Current Flashback Database Window

At any given time, the earliest point in time to which you can actually rewind your database by using Flashback Database can be determined by querying the V$FLASHBACK_DATABASE_LOG view as shown in this example:

```
SELECT OLDEST_FLASHBACK_SCN, OLDEST_FLASHBACK_TIME
      FROM V$FLASHBACK_DATABASE_LOG;
```

If the results of this query indicate that you cannot reach your intended flashback retention target, increase the size of your flash recovery area to accomodate more flashback logs than value indicated by V$FLASHBACK_DATABASE_LOG.ESTIMATED_FLASHBACK_SIZE. Also, when you are deciding whether to use Flashback Database instead of point-in-time recovery, this value will tell you whether you can reach your desired target time before you start the Flashback Database operation.

Performance Tuning for Flashback Database

Maintaining flashback logs imposes comparatively limited overhead on an Oracle database instance. Changed blocks are written from memory to the flashback logs at relatively infrequent, regular intervals, to limit processing and I/O overhead.

To achieve good performance for large production databases with Flashback Database enabled, Oracle Corporation recommends the following:

- Use a fast file system for your flash recovery area, preferably without operating system file caching. Files the database creates in the flash recovery area, including flashback logs, are typically large. Operating system file caching is typically not effective for these files, and may actually add CPU overhead for reading from and writing to these files. Thus, it is recommended to use a file system that avoids operating system file caching, such as ASM, or the Solaris 2.8 file system with direct I/O.
- Configure enough disk spindles for the file system that will hold the flash recovery area. For large production databases, multiple disk spindles may be needed to support the required disk throughput for the database to write the flashback logs effectively.
- If the storage system used to hold the flash recovery area does not have non-volatile RAM, try to configure the file system on top of striped storage volumes, with a relatively small stripe size such as 128K. This will allow each write to the flashback logs to be spread across multiple spindles, improving performance
- For large, production databases, set the init.ora parameter LOG_BUFFER to be at least 8MB. This makes sure the database allocates maximum memory (typically 16MB) for writing flashback database logs.

The overhead of turning on logging for Flashback Database depends on the read-write mix of the database workload. The more write-intensive the workload, the higher the overhead caused by turning on logging for Flashback Database. (Queries do not change data and thus do not contribute to logging activity for Flashback Database.)

Monitoring Flashback Database

The best way to monitor system usage due to flashback logging is to take performance statistics using the Oracle Statspack. For example, if you see "`flashback buf free by RVWR`" as the top wait event, it indicates that Oracle cannot write flashback logs very quickly. In such a case, you may want to tune the file system and storage used by the flash recovery area, possibly using one of the methods described in "Performance Tuning for Flashback Database" on page 9-19.

The V$FLASHBACK_DATABASE_STAT view (described in *Oracle Database Reference*) shows the bytes of flashback data logged by the database. Each row in the view shows the statistics accumulated (typically over the course of an hour). The FLASHBACK_DATA and REDO_DATA columns describe bytes of flashback data and redo data written respectively during the time interval, while the DB_DATA column describe bytes of data blocks read and written. Note that FLASHBACK_DATA and

REDO_DATA correspond to sequential writes, while DB_DATA corresponds to random reads and writes.

Because of the difference between sequential I/O and random I/O, a better indication of I/O overhead is the number of I/O operations issued for flashback logs. The following statistics in V$SYSSTAT can tell you the number of I/O operations your instance has issued for various purposes:

Column Name	Column Meaning
Physical write I/O request	The number of write operations issued for writing data blocks
physical read I/O request	The number of read operations issued for reading data blocks
redo writes	The number of write operations issued for writing to the redo log.
flashback log writes	The number of write operations issued for writing to flashback logs.

See *Oracle Database Reference* for more details on columns in the V$SYSSTAT view.

Running the FLASHBACK DATABASE Command from RMAN

1. Query the target database to determine the range of possible Flashback Database SCNs. The following queries show you the the latest and earliest SCN in the flashback window:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE;

SQL> SELECT OLDEST_FLASHBACK_SCN, OLDEST_FLASHBACK_TIME
        FROM V$FLASHBACK_DATABASE_LOG;
```

2. Start RMAN and connect to the target database. For example:

```
rman TARGET /
```

3. Run the FLASHBACK DATABASE command to return the database to a prior TIME, SCN, or archived log SEQUENCE number. If you configured sbt channels, RMAN automatically restores archived logs from tape as needed during the Flashback Database operation. For example:

```
RMAN> FLASHBACK DATABASE TO SCN 46963;
```

```
RMAN> FLASHBACK DATABASE TO SEQUENCE 5304;

RMAN> FLASHBACK DATABASE TO TIME (SYSDATE-1/24);

RMAN> FLASHBACK DATABASE TO TIME timestamp('2002-11-05 14:00:00');

RMAN> FLASHBACK DATABASE
  TO TIME to_timestamp('2002-11-11 16:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

When the Flashback Database operation completes, you can evaluate the results by opening the database read-only and run some queries to check whether your Flashback Database has returned the database to the desired state.

```
RMAN> SQL 'ALTER DATABASE OPEN READ ONLY';
```

At this point you have several options:

- If you are content with the results, you can make the database available by performing an ALTER DATABASE OPEN RESETLOGS.

  ```
  RMAN> ALTER DATABASE OPEN RESETLOGS
  ```

- If you discover that you have chosen the wrong target time for your Flashback Database operation, you can use RECOVER DATABASE UNTIL to bring the database forward, or perform FLASHBACK DATABASE again with an SCN further in the past. You can completely undo the effects of your flashback operation by performing complete recovery of the database:

  ```
  RMAN> RECOVER DATABASE;
  ```

- If you only want to retrieve some lost data from the past time, you can open the database read-only, then perform a logical export of the data using an Oracle export utility (Data Pump Export or Original Export), then run RECOVER DATABASE to return the database to the present time and re-import the data using the Oracle import utility that corresponds to the export utility you used.

Note that, as with point-in-time recovery, you lose all updates to the database after the target SCN for the Flashback Database operation.

Running the FLASHBACK DATABASE Command from SQL*Plus

The FLASHBACK DATABASE command in SQL*Plus takes essentially the same options and performs essentially the same behavior as FLASHBACK DATABASE as performed in RMAN. The chief difference is that RMAN, being aware of backups of your database files, can restore from backup automatically any needed archived

logs required for the Flashback Database process. When using FLASHBACK DATABASE in SQL*Plus, all files required to complete the operation must already be present on disk.

Using Oracle Flashback Features Together in Data Recovery: Scenario

The following scenario shows how one might use the various flashback features of Oracle (Oracle Flashback Query, Oracle Flashback Transaction Query, Oracle Flashback Table, and Oracle Flashback Database) to recover from a data loss due to a user or application error.

At 17:00 an HR administrator discovers that an employee "JOHN" is missing from the EMPLOYEE table. This employee was present in the table at 14:00, when she last checked. This means that someone accidentally deleted "JOHN" from the table between 14:00 and 17:00. The HR administrator re-inserts the missing employee row into the EMPLOYEE table with the following use of Flashback Query:

```
INSERT INTO employee
    SELECT * FROM employee AS OF TIMESTAMP
        TO_TIMESTAMP('2003-04-04 14:00:00', 'YYYY-MM-DD HH:MI:SS')
        WHERE name = 'JOHN';
```

She can find out more information about when "JOHN" was deleted, the transaction which deleted "JOHN", and the user who deleted "JOHN" by using Flashback Version Query and Flashback Transaction Query as follows:

```
SELECT commit_timestamp , logon_user FROM FLASHBACK_TRANSACTION_QUERY
     WHERE xid IN
        (SELECT versions_xid FROM employee VERSIONS BETWEEN
         TIMESTAMP TO_TIMESTAMP('2003-04-04 14:00:00', 'YYYY-MM-DD HH:MI:SS')
         and TO_TIMESTAMP('2003-04-04 17:00:00', 'YYYY-MM-DD HH:MI:SS')
         WHERE name = 'JOHN');
```

If at this time she discovers many other logical data errors in the EMPLOYEE table, she can recover the whole table to the state at 14:00 by using Flashback Table:

```
FLASHBACK TABLE employee TO TIMESTAMP TO_TIMESTAMP('2003-04-04 14:00:00',
'YYYY-MM-DD HH:MI:SS');
```

Finally, if many other tables also contain errors due to transactions during the same interval, flashback database can return the entire database to its state before the errors:

```
FLASHBACK DATABASE
  TO TIME to_timestamp('2003-04-04 14:00:00', 'YYYY-MM-DD HH:MI:SS');
```

For more examples of using flashback features to recover from user errors, see *Oracle High Availability Architecture and Best Practices.*

10

RMAN Tablespace Point-in-Time Recovery (TSPITR)

Recovery Manager (RMAN) automatic **tablespace point-in-time recovery** (commonly abbreviated **TSPITR**) enables you to quickly recover one or more tablespaces in an Oracle database to an earlier time, without affecting the state of the rest of the tablespaces and other objects in the database.

This chapter explains when you can and cannot use TSPITR, what RMAN actually does to your database during TSPITR, how to prepare a database for TSPITR, how to run TSPITR, and options for controlling the TSPITR process.

This chapter contains the following sections:

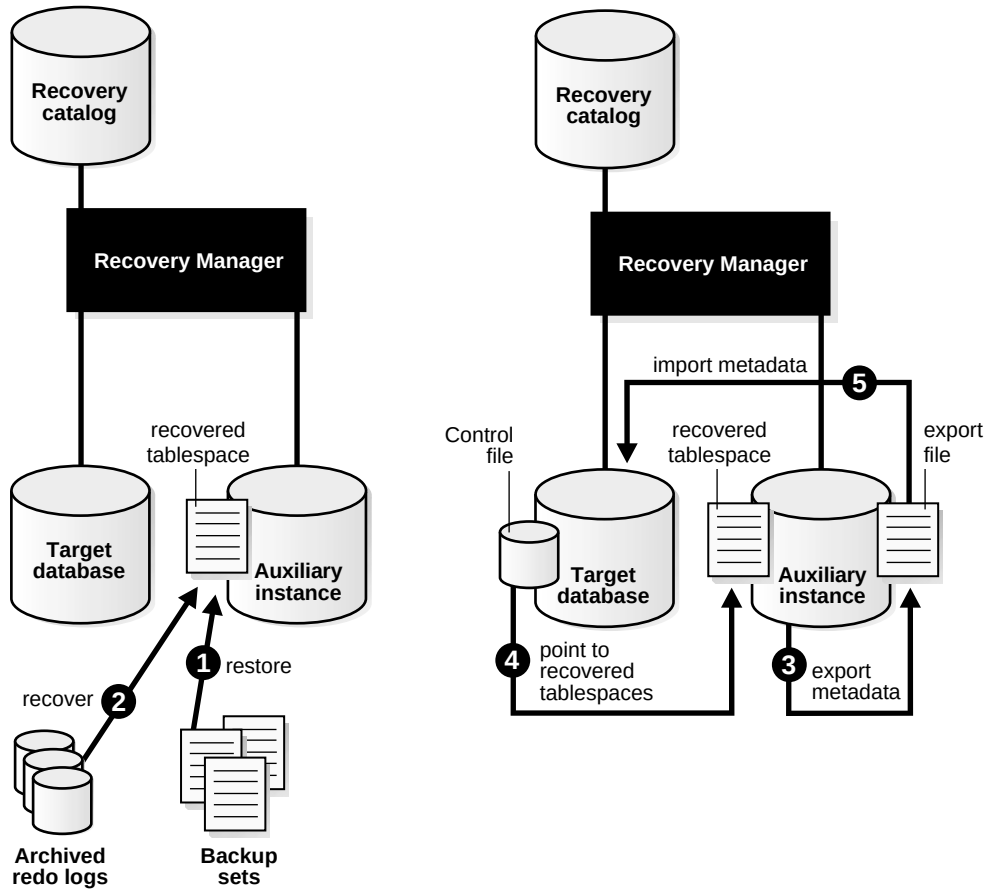
Understanding RMAN TSPITR

In order to use TSPITR effectively, you need to understand what problems it can solve for you, what the major elements used in TSPITR are, what RMAN does during TSPITR, and limitations on when and how it can be applied.

RMAN TSPITR Concepts

Figure 10–1 illustrates the context within which TSPITR takes place, and a general outline of the process.

Figure 10–1 Tablespace Point-in-Time Recovery (TSPITR) Architecture



The figure contains the following entities:

- The **target instance**, containing the tablespace to be recovered
- The **Recovery Manager** client

- The **control file** and (optional) **recovery catalog**, used for the RMAN repository records of backup activity
- **Archived redo logs** and **backup sets** from the target database, which are the source of the reconstructed tablespace.
- The **auxiliary instance**, an Oracle database instance used in the recovery process to perform the actual work of recovery.

There are four other important terms related to TSPITR, which will be used in the rest of this discussion:

- The **target time**, the point in time or SCN that the tablespace will be left at after TSPITR
- The **recovery set**, which consists of the datafiles containing the tablespaces to be recovered;
- The **auxiliary set**, which includes datafiles required for TSPITR of the recovery set which are not themselves part of the recovery set. The auxiliary set typically includes:
  - A copy of the SYSTEM tablespace
  - Datafiles containing rollback or undo segments from the target instance
  - In some cases, a temporary tablespace, used during the export of database objects from the auxiliary instance

  The auxiliary instance has other files associated with it, such as a control file, parameter file, and online logs , but they are not part of the auxiliary set.
- The **auxiliary destination**, an optional location on disk which can be used to store any of the auxiliary set datafiles, control files and online logs of the auxiliary instance during TSPITR. Files stored here can be deleted after TSPITR is complete.

All of these terms will be referenced throughout the remainder of this chapter.

How TSPITR Works With an RMAN-Managed Auxiliary Instance

To perform TSPITR of the recovery set using RMAN and an automated auxiliary instance, you carry out the preparations for TSPITR described in "Planning and Preparing for TSPITR" on page 10-6, and then issue the RECOVER TABLESPACE command, specifying, at a minimum, the tablespaces of the recovery set and the target time for the point-in-time recovery, and, if desired, an auxiliary destination as well.

RMAN then carries out the following steps:

1. If there is no connection to an auxiliary instance, RMAN creates the auxiliary instance, starts it up and connects to it.
2. Takes the tablespaces to be recovered offline in the target database
3. Restores a backup controlfile from a point in time before the target time to the auxiliary instance
4. Restores the datafiles from the recovery set and the auxiliary set to the auxiliary instance. Files are restored either in locations you specify for each file, or the original location of the file (for recovery set files) or in the auxiliary destination (for auxiliary set files, if you used the `AUXILIARY DESTINATION` argument of `RECOVER TABLESPACE`)
5. Recovers the restored datafiles in the auxiliary instance to the specified time
6. Opens the auxiliary database with the `RESETLOGS` option
7. Exports the dictionary metadata about objects in the recovered tablespaces to the target database
8. Shuts down the auxiliary instance
9. Issues `SWITCH` commands on the target instance, so that the target database control file now points to the datafiles in the recovery set that were just recovered at the auxiliary instance.
10. Imports the dictionary metadata from the auxiliary instance to the target instance, allowing the recovered objects to be accessed.
11. Deletes all auxiliary set files.

At that point the TSPITR process is complete. The recovery set datafiles are returned to their contents at the specified point in time, and belong to the target database.

Deciding When to Use TSPITR

Like a table import, RMAN TSPITR enables you to recover a consistent data set; however, the data set recovered includes an entire tablespace rather than one object.

RMAN TSPITR is most useful for situations such as these:

- Recovering data lost after an erroneous `TRUNCATE TABLE` statement;
- Recovering from logical corruption of a table;

- Undoing the effects of an incorrect batch job or other DML statement that has affected only a subset of the database;
- Recovering a logical schema to a point different from the rest of the physical database, when multiple schemas exist in separate tablespaces of one physical database.

Note that, as with database point-in-time recovery (DBPITR), you cannot perform TSPITR if you do not have your archived redo logs. For databases running in NOARCHIVELOG mode, you cannot perform TSPITR.

Limitations of TSPITR

There are a number of situations which you cannot resolve by using TSPITR.

- You cannot recover dropped tablespaces.
- You cannot recover a renamed tablespace to a point in time before it was renamed. If you try to perform a TSPITR to an SCN earlier than the rename operation, RMAN cannot find the new tablespace name in the repository as of that earlier SCN (because the tablespace did not have that name at that SCN).

  In this situation, you must recover the entire database to a point in time before the tablespace was renamed. The tablespace will be found under the name it had at that earlier time.
- You cannot recover tables without their associated constraints, or constraints without the associated tables.
- You cannot use TSPITR to recover any of the following:
  - Replicated master tables
  - Partial tables (for example, if you perform RMAN TSPITR on partitioned tables and spread partitions across multiple tablespaces, then you must recover all tablespaces which include partitions of the table.)
  - Tables with VARRAY columns, nested tables, or external files
  - Snapshot logs and snapshot tables
  - Tablespaces containing undo or rollback segments
  - Tablespaces that contain objects owned by SYS, including rollback segments

TSPITR has some other limitations:

- If a datafile was added after the point to which RMAN is recovering, an empty datafile by the same name will be included in the tablespace after RMAN TSPITR.
- TSPITR will not recover query optimizer statistics for recovered objects. You must gather new statistics after the TSPITR.
- Assume that you run TSPITR on a tablespace, and then bring the tablespace online at time *t*. Backups of the tablespace created before time *t* are no longer usable for recovery with a current control file. You cannot run TSPITR again on this tablespace to recover it to any time less than or equal to time *t*, nor can you use the current control file to recover the database to any time less than or equal to *t*. Therefore, you must back up the tablespace as soon as TSPITR is complete.

Limitations of TSPITR Without a Recovery Catalog If you do not use a recovery catalog when performing TSPITR, then note the following special restrictions:

- The undo segments at the time of the TSPITR must be part of the auxiliary set. Because RMAN has no historical record of the undo in the control file, RMAN assumes that the current rollback or undo segments were the same segments present at the time to which recovery is performed. If the undo segments have changed since that time, then TSPITR will fail.
- TSPITR to a time that is too old may not succeed if Oracle has reused the control file records for needed backups. (In planning your database, set the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter to a value large enough to ensure that control file records needed for TSPITR are kept.)
- When not using a recovery catalog, the current control file has no record of the older incarnation of the recovered tablespace. Thus, recovery with a current control file that involves this tablespace can no longer use a backup taken prior to time *t*. You can, however, perform incomplete recovery of the whole database to any time less than or equal to *t*, if you can restore a backup control file from before time *t*.

Planning and Preparing for TSPITR

There are several steps to be carried out in preparing for TSPITR:

- Choosing the Right Target Time for TSPITR
- Determining the Recovery Set: Analyzing Data Relationships
- Identifying and Preserving Objects That Will Be Lost After TSPITR

Choosing the Right Target Time for TSPITR

It is extremely important that you choose the right target time or SCN for your TSPITR. As noted already, once you bring a tablespace online after TSPITR, you cannot use any backup from a time earlier than the moment you brought the tablespace online. In practice, this means that you cannot make a second attempt at TSPITR if you choose the wrong target time the first time, unless you are using a recovery catalog. (If you have a recovery catalog, however, you can perform repeated TSPITRs to different target times.)

For example, assume that you run TSPITR on a tablespace, and then bring the tablespace online at 5PM on Friday. Backups of the tablespace created before 5PM Friday are no longer usable for recovery with a current control file. You cannot run TSPITR again on this tablespace with a target time earlier than 5PM Friday, nor can you use the current control file to recover the database to any time earlier than 5PM Friday. Your only option will be point-in-time recovery of your entire database using a restored control file.

To investigate past states of your data to identify the target time for TSPITR, you can use features of Oracle such as Oracle Flashback Query, Oracle Transaction Query and Oracle Flashback Version Query to find the point in time when unwanted database changes occurred. See "Oracle Flashback Query: Recovering at the Row Level" on page 9-3 for more details on Flashback Query, and *Oracle Database Application Developer's Guide - Fundamentals* for more information on Flashback Transaction Query and Flashback Version Query.

Determining the Recovery Set: Analyzing Data Relationships

Your recovery set starts out including the datafiles for the tablespaces you wish to recover. If, however, objects in the tablespaces you need have relationships (such as constraints) to objects in other tablespaces, you will have to account for this relationship before you can perform TSPITR. You have three choices when faced with such a relationship:

- Add the tablespace including the related objects to your recovery set
- Remove the relationship
- Suspend the relationship for the duration of TSPITR

Identifying and Resolving Dependencies on the Primary Database

The `TS_PITR_CHECK` view lets you identify relationships between objects that span the recovery set boundaries. If this view returns rows when queried, then

investigate and correct the problem. Proceed with TSPITR *only* when TS_PITR_CHECK view returns no rows for the tablespaces not in the recovery set. Record all actions performed during this step so that you can re-create any suspended or removed relationships after completing TSPITR.

The following query illustrates how to use the TS_PITR_CHECK view. For an example with an initial recovery set consisting of tools and users, the SELECT statement against TS_PITR_CHECK would be as follows:

```
SELECT *
FROM SYS.TS_PITR_CHECK
WHERE (
        TS1_NAME IN ('USERS','TOOLS')
        AND TS2_NAME NOT IN ('USERS','TOOLS')
      )
OR    (
        TS1_NAME NOT IN ('USERS','TOOLS')
        AND TS2_NAME IN ('USERS','TOOLS')
      );
```

To run a complete TSPITR check on all the tablespaces in the database (not just the tablespaces in the recovery set), you can run the following query:

```
SELECT *
FROM SYS.TS_PITR_CHECK
WHERE (
         'SYSTEM' IN (TS1_NAME, TS2_NAME)
         AND TS1_NAME <> TS2_NAME
         AND TS2_NAME <> '-1'
      )
OR    (
         TS1_NAME <> 'SYSTEM'
         AND TS2_NAME = '-1'
      );
```

Because of the number and width of the columns in the TS_PITR_CHECK view, you may want to format the columns as follows when running the query:

```
SET LINESIZE 120
COLUMN OBJ1_OWNER HEADING "own1"
COLUMN OBJ1_OWNER FORMAT a6
COLUMN OBJ1_NAME HEADING "name1"
COLUMN OBJ1_NAME FORMAT a5
COLUMN OBJ1_SUBNAME HEADING "subname1"
COLUMN OBJ1_SUBNAME FORMAT a8
COLUMN OBJ1_TYPE HEADING "obj1type"
```

```
COLUMN OBJ1_TYPE FORMAT a8 word_wrapped
COLUMN TS1_NAME HEADING "ts1_name"
COLUMN TS1_NAME FORMAT a6
COLUMN OBJ2_NAME HEADING "name2"
COLUMN OBJ2_NAME FORMAT a5
COLUMN OBJ2_SUBNAME HEADING "subname2"
COLUMN OBJ2_SUBNAME FORMAT a8
COLUMN OBJ2_TYPE HEADING "obj2type"
COLUMN OBJ2_TYPE FORMAT a8 word_wrapped
COLUMN OBJ2_OWNER HEADING "own2"
COLUMN OBJ2_OWNER FORMAT a6
COLUMN TS2_NAME HEADING "ts2_name"
COLUMN TS2_NAME FORMAT a6
COLUMN CONSTRAINT_NAME HEADING "cname"
COLUMN CONSTRAINT_NAME FORMAT a5
COLUMN REASON HEADING "reason"
COLUMN REASON FORMAT a25 word_wrapped
```

Assume a case in which the partitioned table `tp` has two partitions, `p1` and `p2`, that exist in tablespaces `users` and `tools` respectively. Also assume that a partitioned index called `tpind` is defined on `tp`, and that the index has two partitions `id1` and `id2` (that exist in tablespaces `id1` and `id2` respectively). In this case, you would get the following output when `TS_PITR_CHECK` is queried against tablespaces `users` and `tools` (assuming appropriate formatting):

```
own1   name1 subname1 obj1type ts1_name name2 subname2 obj2type own2      ts2_name   cname reason
---    ----  -----    ------   -------  ----  ------   -------- ---       --------   ---   ------
SYSTEM  TP   P1       TABLE    USER     TPIND IP1      INDEX    PARTITION PARTITION  SYS   ID1 Partitioned
Objects not fully contained in the recovery set
SYSTEM  TP   P2       TABLE    TOOLS    TPIND IP2      INDEX    PARTITION PARTITION  SYS   ID2 Partitioned
Objects not fully contained in the recovery set
```

The table `SYSTEM.tp` has a partitioned index `tpind` that consists of two partitions, `ip1` in tablespace `id1` and `ip2` in tablespace `id2`. To perform TSPITR, you must either drop `tpind` or include `id1` and `id2` in the recovery set.

> **See Also:** *Oracle Database Reference* for more information about the `TS_PITR_CHECK` view

Identifying and Preserving Objects That Will Be Lost After TSPITR

When RMAN TSPITR is performed on a tablespace, any objects created after the target recovery time are lost. You can preserve such objects, once they are identified, by exporting them before TSPITR using an Oracle export utility (Data Pump Export

or Original Export) and re-importing them afterwards using the corresponding import utility.

To see which objects will be lost in TSPITR, query the TS_PITR_OBJECTS_TO_BE_DROPPED view on the primary database. The contents of the view are described in Table 10–1.

Table 10–1 TS_PITR_OBJECTS_TO_BE_DROPPED View

Column Name	Meaning
OWNER	Owner of the object to be dropped.
NAME	The name of the object that will be lost as a result of undergoing TSPITR
CREATION_TIME	Creation timestamp for the object.
TABLESPACE_NAME	Name of the tablespace containing the object.

Filter the view for objects whose CREATION_TIME is after the target time for TSPITR. For example, with a recovery set consisting of users and tools, and a recovery point in time of November 2, 2002, 7:03:11 AM, issue the following statement:

```
SELECT OWNER, NAME, TABLESPACE_NAME,
       TO_CHAR(CREATION_TIME, 'YYYY-MM-DD:HH24:MI:SS')
       FROM TS_PITR_OBJECTS_TO_BE_DROPPED
WHERE TABLESPACE_NAME IN ('USERS','TOOLS')
AND CREATION_TIME > TO_DATE('02-NOV-02:07:03:11','YY-MON-DD:HH24:MI:SS')
ORDER BY TABLESPACE_NAME, CREATION_TIME;
```

(The TO_CHAR and TO_DATE functions are used to avoid issues with different national date formats. You can, of course, use local date formats in your own work.)

> **See Also:** *Oracle Database Reference* for more information about the TS_PITR_OBJECTS_TO_BE_DROPPED view

Performing Basic RMAN TSPITR

Having selected your tablespaces to recover and your target time, you are now ready to perform RMAN TSPITR. You have a few different options available to you:

- **Fully automated TSPITR**--in which you specify an auxiliary destination and let RMAN manage all aspects of the TSPITR. This is the simplest way to perform TSPITR, and is recommended unless you specifically need more control over

the location of recovery set files after TSPITR or auxiliary set files during TSPITR, or control over the channel configurations or some other aspect of your auxiliary instance.

- **Customized TSPITR with an automatic auxiliary instance**--in which you base your TSPITR on the behavior of fully automated TSPITR, possibly still using an auxiliary destination, but customize one or more aspects of the behavior, such as the location of auxiliary set or recovery set files, or specifying initialization parameters or channel configurations for the auxiliary instance created and managed by RMAN.
- **TSPITR with your own auxiliary instance**--in which you take responsibility for setting up, starting, stopping and cleaning up the auxiliary instance used in TSPITR, and possibly also manage the TSPITR process using some of the methods available in customized TSPITR with an automatic auxiliary instance.

Fully Automated RMAN TSPITR

When performing fully automated TSPITR, letting RMAN manage the entire process, there are only two requirements beyond the preparations in "Planning and Preparing for TSPITR" on page 10-6:

- You must specify the auxiliary destination for RMAN to use for the auxiliary set datafiles and other files for the auxiliary instance.
- You must configure any channels required for the TSPITR on the target instance. (The auxiliary instance will use the same channel configuration as the target instance when performing the TSPITR.)

RMAN bases as much of the configuration for TSPITR as possible on your target database. During TSPITR, the recovery set datafiles are written in their current locations on the target database. The same channel configurations in effect on the target database are used on the auxiliary instance when restoring files from backup. Auxiliary set datafiles and other auxiliary instance files, however, are stored in the auxiliary destination.

Using an Auxiliary Destination

Oracle Corporation recommends that you use an auxiliary destination with your auxiliary instance. Even if you use other methods to rename some or all of the auxiliary set datafiles, specifying an `AUXILIARY DESTINATION` parameter provides a default location for auxiliary set datafiles for which names are not specified. This way, TSPITR will not fail if you inadvertently do not provide names for all auxiliary set datafiles.

To specify an auxiliary destination, find a location on disk where there is enough space to hold your auxiliary set datafiles. Then, use the AUXILIARY DESTINATION parameter in your RECOVER TABLESPACE command to specify the auxiliary destination location, as shown in the next section.

Performing Fully Automated RMAN TSPITR

To actually peform automated RMAN TSPITR, start the RMAN client, connecting to the target database and, if applicable, a recovery catalog. This example shows connecting in NOCATALOG mode, using operating system authentication:

```
% rman TARGET /
```

> **Note:** Do not connect to an auxiliary instance when starting the RMAN client for automated TSPITR. If there is no connected auxiliary instance, RMAN constructs the automatic auxiliary instance for you when carrying out the RECOVER TABLESPACE command. (If there is a connected auxiliary instance, RMAN will assume that you are trying to manage your own auxiliary instance, and try to use the connected auxiliary for TSPITR.)

If you have configured channels that RMAN can use to restore from backup on the primary instance, then you are ready to perform TSPITR now, by running the RECOVER TABLESPACE... UNTIL... command.

This example returns the users and tools tablespaces to the end of log sequence number 1300, and stores the auxiliary instance files (including auxiliary set datafiles) in the destination /disk1/auxdest:

```
RMAN> RECOVER TABLESPACE users, tools
     UNTIL LOGSEQ 1300 THREAD 1
      AUXILIARY DESTINATION '/disk1/auxdest';
```

Assuming the TSPITR process completes without error, the tablespaces are taken offline by RMAN, restored from backup and recovered to the desired point in time on the auxiliary instance, and then re-imported to the target database. The tablespaces are left offline at the end of the process. All auxiliary set datafiles and other auxiliary instance files are cleaned up from the auxiliary destination.

Tasks to Perform After Successful TSPITR

If TSPITR completes successfully, you must back up the recovered tablespaces, and then you can bring them online.

Backing Up Recovered Tablespaces After TSPITR It is very important that you backup recovered tablespaces immediately after TSPITR is completed.

After you perform TSPITR on a tablespace, you cannot use backups of that tablespace from before the TSPITR was completed and the tablespace put back on line. If you start using the recovered tablespaces without taking a backup, you are running your database without a usable backup of those tablespaces. For this example, the users and tools tablespaces must be backed up, as follows:

```
RMAN> BACKUP TABLESPACE users, tools;
```

You can then safely bring the tablespaces online, as follows:

```
RMAN> SQL "ALTER TABLESPACE users, tools ONLINE";
```

Your recovered tablespaces are now ready for use.

Handling Errors in Automated TSPITR

In the event of an error during automated TSPITR, you should refer to "Troubleshooting RMAN TSPITR" on page 10-29. The auxiliary set datafiles and other auxiliary instance files will be left in place in the auxililary destination as an aid to troubleshooting. The state of the recovery set files is determined by the type of failure. Once you resolve the problem, you can try your TSPITR operation again.

Performing Customized RMAN TSPITR with an RMAN-Managed Auxiliary Instance

There are several aspects of RMAN TSPITR which you can customize while still mostly following the basic procedure described in "Fully Automated RMAN TSPITR" on page 10-11:

- Renaming or relocating your recovery set datafiles, so that the datafiles making up the recovered tablespaces are not stored in the original locations after TSPITR
- Specifying a location other than the auxiliary destination for some or all auxiliary set datafiles (or not using an auxiliary destination at all)
- Setting up image copy backups of your datafiles in advance, to speed up TSPITR by avoiding restores from backup
- Using a different channel configuration for the auxiliary instance

- Specifying different initialization parameters for your RMAN-managed auxiliary instance

Renaming TSPITR Recovery Set Datafiles with SET NEWNAME

You may not want the recovery set datafiles restored and recovered in their original locations. The SET NEWNAME command, used in a RUN block, lets you specify a new destination for the restore from backup and recovery of a datafile.

> **Note:** CONFIGURE AUXNAME can be used to rename recovery set datafiles as well, but the effects of doing so are quite different and the two commands cannot be used interchangeably. (They do interact, in that if you use SET NEWNAME to rename a file, this takes precedence over any renaming performed with CONFIGURE AUXNAME.) Refer to the discussion of "Using Image Copies for Faster TSPITR Performance" on page 10-18 for details.

Create a RUN block and use SET NEWNAME commands within it to specify new recovery set filenames, as shown here:

```
RUN {
...
   SET NEWNAME FOR DATAFILE 'ORACLE_HOME/oradata/trgt/users01.dbf'
      TO '/newfs/users01.dbf';
   ...other setup commands...
   RECOVER TABLESPACE users, tools UNTIL SEQUENCE 1300 THREAD 1;
}
```

RMAN restores the specified datafile from backup to the new location during TSPITR and recovers it in the new location, and updates the control file so that the newly recovered datafile replaces the old one in the control file. Any existing image copy backup of a datafile found at the new specified location is overwritten.

If the name specified with SET NEWNAME conflicts with the name of a valid datafile in the target database, then RMAN reports an error while executing the RECOVER command. The valid datafile is not overwritten.

Note that RMAN does not detect conflicts between names set with SET NEWNAME and current datafile names on the target database until the actual RECOVER TABLESPACE... UNTIL operation. At that point, the conflict is detected, TSPITR fails and RMAN reports an error. If you rename your recovery set datafiles, be sure

to assign them names that do not conflict with each other, or with the names of your current datafiles.

Renaming TSPITR Auxiliary Set Datafiles

Unlike the recovery set datafiles, which can be and usually are stored in their original locations, the auxiliary set datafiles must not overwrite the corresponding original files in the target database. If you do not specify a location for an auxiliary set file that is different from its original location, then TSPITR will fail when RMAN attempts to overwrite the corresponding file in the original database and discover that the file is in use.

The simplest way to provide locations for your auxiliary set datafiles is to specify an auxiliary destination for TSPITR. However, RMAN supports two other methods of controlling the location of your auxiliary set datafiles: specifying new names for individual files with SET NEWNAME, and using DB_FILE_NAME_CONVERT to provide rules for converting datafile names in the target database to datafile names for the auxiliary database.

Even if you intend to use either of these methods to provide locations for specific files, it is still suggested that you provide an AUXILIARY DESTINATION argument to RECOVER TABLESPACE. This will ensure that, if you overlook renaming some auxiliary set datafiles, your TSPITR will still succeed. Any files not otherwise renamed will be placed in the auxiliary destination.

Renaming TSPITR Auxiliary Set Datafiles with SET NEWNAME

To use the SET NEWNAME command to specify a new name for an auxiliary set datafile, enclose your RECOVER TABLESPACE command in a RUN block, and use a SET NEWNAME command within the RUN block to rename the file. For example:

```
RMAN> RUN
{
  SET NEWNAME FOR DATAFILE '?/oradata/prod/system01.f'
    TO '/disk1/auxdest/system01.f'
  RECOVER TABLESPACE users, tools
     UNTIL LOGSEQ 1300 THREAD 1
     AUXILIARY DESTINATION '/disk1/auxdest';
}
```

The resulting behavior depends upon whether there is a file at /disk1/auxdest/system01.f when the RECOVER TABLESPACE command is executed. If there is an image copy backup of the file ?/oradata/system01.f at the specified location, created at an SCN prior to the target time for TSPITR, then

the behavior is as described in "SET NEWNAME and CONFIGURE AUXNAME With Auxiliary Set Image Copies" on page 10-19. Otherwise, the auxiliary set datafile will be restored to the NEWNAME specified instead of the default location. If your intention is only to control where the auxiliary set datafiles are stored, you should make sure that there is no file stored at the location specified by SET NEWNAME before performing your TSPITR.

Using DB_FILE_NAME_CONVERT to Name Auxiliary Set Datafiles

If you do not want to use an auxiliary destination for all of your auxiliary set datafiles, but you also do not want to name every file individually, you can include a DB_FILE_NAME_CONVERT initialization parameter in the initialization parameter file used by your auxiliary instance. You can only use this method in two circumstances:

- If you are creating your own initialization parameter file for RMAN's automatically managed auxiliary instance, as described in "Customizing Initialization Parameters for the Automatic Auxiliary Instance" on page 10-21;
- If you are creating your own auxiliary instance, as described in "Performing RMAN TSPITR Using Your Own Auxiliary Instance" on page 10-22.

Refer to the appropriate discussion for your circumstance, to see how to add a parameter to your initialization parameter file.

The DB_FILE_NAME_CONVERT parameter in the auxiliary instance specifies how to derive names for files in the auxiliary instance from the original names of the corresponding files in the target instance.

For example, assume that the target instance contains the following files:

- ?/oradata/trgt/system01.dbf of the SYSTEM tablespace
- ?/oradata/trgt/undotbs01.dbf of the undotbs tablespace

and you need to locate the corresponding files in the auxiliary instance in '/bigtmp', then you would add the following line to the auxiliary instance parameter file:

```
DB_FILE_NAME_CONVERT=('?/oradata/trgt', '/bigtmp')
```

The most important thing to remember is that DB_FILE_NAME_CONVERT needs to be present in the auxiliary instance parameter file.

If the auxiliary instance was manually created, add DB_FILE_NAME_CONVERT to the auxiliary instance parameter file (wherever it resides).

Note that you can still rename individual auxiliary set datafiles using SET NEWNAME or CONFIGURE AUXNAME. Also, files that do not match the patterns provided in DB_FILE_NAME_CONVERT will not be renamed. You may wish to use the AUXILIARY DESTINATION parameter of RECOVER TABLESPACE to ensure that all auxiliary set datafiles are sent to some destination. If a file is not renamed at all, TSPITR will fail.

Order of Precedence Among File Renaming Methods

The different methods of renaming files follow an order of precedence, as follows:

- SET NEWNAME
- CONFIGURE AUXNAME
- DB_FILE_NAME_CONVERT
- AUXILIARY DESTINATION argument to RECOVER TABLESPACE

Settings higher on the list override settings lower on the list, in situations where both have been applied (by, for example, running RECOVER TABLESPACE... AUXILIARY DESTINATION on a target database where some auxiliary set datafiles also have auxnames configured with CONFIGURE AUXNAME).

Specifying Auxiliary Instance Control File Location

You can specify your own location for the control file of your auxiliary instance, if you use a client-side initialization parameter file. Set the CONTROL_FILES initialization parameter to specify any location you wish for the control files.

If you do not explicitly specify a location for the control file, RMAN will locate it in the auxiliary destination if you use the AUXILIARY DESTINATION parameter when performing TSPITR. If you do not use an AUXILIARY DESTINATION parameter, the auxiliary instance control files are stored in an operating system-specific location. (on Unix, *ORACLE_HOME*/rdbms/admin/params_auxinit.ora).

No matter where you store your auxiliary instance control file, it is removed at the end of a successful TSPITR operation. Because control files are relatively small, it is rare that RMAN will encounter a problem creating an auxiliary control file, but if there is not enough space to create the control file, TSPITR will fail.

Specifying Auxiliary Instance Online Log Location

If you specify the LOG_FILE_NAME_CONVERT initialization parameter in your auxiliary instance parameter file, this parameter will determine the online redo log

location. Otherwise, if RMAN is using an auxiliary destination and managing the auxiliary instance for you, it creates the online redo log in the auxiliary destination.

> **Note:** If you do not specify a location for the online redo logs using LOG_FILE_NAME_CONVERT or AUXILIARY DESTINATION, your TSPITR will fail trying to create the online redo logs. Even if DB_FILE_CREATE_DEST or LOG_FILE_CREATE_DEST are specified in the initialization parameter file, in TSPITR they do not control the creation of the online redo logs of the auxiliary instance.

Using Image Copies for Faster TSPITR Performance

TSPITR performance can be greatly enhanced by redirecting RMAN to use existing image copies of the recovery set and auxiliary set datafiles on disk in TSPITR, rather than restoring them from backup. You can use the CONFIGURE AUXNAME command with image copies of recovery set datafiles or auxiliary set datafiles, or the SET NEWNAME command with image copies of auxiliary set datafiles, to tell RMAN about the possible existence of an image copy of a datafile.

While exact details vary depending on the command used and whether the file is an auxiliary set or recovery set file, in general, if a suitable image copy is available in the specified location, then during TSPITR, RMAN uncatalogs the image copy from the RMAN repository of the target instance, and catalogs it in the control file of the auxiliary instance. The auxiliary instance then performs point-in-time recovery using the image copy.

Details of using image copies with each type of file are explained in the following sections.

Using CONFIGURE AUXNAME With Recovery Set Image Copies

During TSPITR, RMAN looks in the specified AUXNAME location for the datafile, to see whether the file there is an image copy backup of the datafile, with a checkpoint SCN early enough that it can be recovered to the target time for TSPITR. If such an image copy is found, it is used in TSPITR. Otherwise, the datafile is restored and recovered in its original location, and any file in the location specified by the AUXNAME is not changed or deleted.

```
RMAN> CONFIGURE AUXNAME FOR DATAFILE 'ORACLE_HOME/oradata/trgt/users01.dbf'
      TO '/newfs/users1.dbf';
...other RMAN commands, if any...
RMAN> RECOVER TABLESPACE users, tools UNTIL SEQUENCE 1300 THREAD 1;
```

`CONFIGURE AUXNAME` is meant to be used as the basis of a strategy to make TSPITR faster by eliminating restore times. If you have tablespaces on which you anticipate performing TSPITR, you can maintain a set of image copies of the affected datafiles, updated periodically to the earliest point to which you expect to perform TSPITR. The expected usage model is:

- Configure the AUXNAME for the file once
- Perform "`BACKUP AS COPY DATAFILE` *n* `FORMAT` *auxname"* regularly to maintain the updated image copy, or use an incrementally updated backups strategy as described in *Oracle Database Backup and Recovery Basics* to keep the image copies up to date without performing full backups of the datafiles
- When TSPITR is needed, specify a target time since the last update of the image copy.

In planning for TSPITR with image copies, remember that you may not know know which tablespaces will require image copies in advance. As discussed in "Determining the Recovery Set: Analyzing Data Relationships" on page 10-7, relationships between the tablespaces you wish to TSPITR and other tablespaces may require that you add tablespaces to your final recovery set, and still other tablespaces may wind up in the auxiliary set. You should configure an AUXNAME for each datafile that is likely to be required, and update image copies of all datafiles often. However, TSPITR will still work if only a subset of datafiles are prepared in advance using this strategy. The process will just take longer, and recover recovery set datafiles for which there are no image copies in their original locations.

Note thatthe order of precedence of naming methods is still respected when you use `CONFIGURE AUXNAME` to rename a recovery set file. A `SET NEWNAME` for a recovery set file will override the effect of the `CONFIGURE AUXNAME` command for the same file. Behavior in this instance will be as described in "Renaming TSPITR Recovery Set Datafiles with SET NEWNAME" on page 10-14. SET NEWNAME used with a recovery set file never refers to an image copy file.

SET NEWNAME and CONFIGURE AUXNAME With Auxiliary Set Image Copies

As with recovery set datafiles, `CONFIGURE AUXNAME` sets a persistent alternative location for an auxiliary set datafile image copy, and `SET NEWNAME` sets an alternative location for the duration of a RUN block. However, RMAN handles values for auxiliary set datafiles differently from recovery set datafiles.

If `SET NEWNAME` is used to specify a new location for an auxiliary set datafile, and there is an image copy at that location with an SCN such that it can be used in

TSPITR, then the image copy will be used. If there is no usable image copy at that location, however, RMAN will restore a usable copy from backup. (If an image copy is present but the SCN is after the target time for TSPITR, then the datafile is overwritten by the restored file.)

If CONFIGURE AUXNAME is used to specify a new location for an auxiliary set datafile, and there is an image copy at that location with an SCN such that it can be used in TSPITR, then the image copy will be used. If there is no usable copy at the specified location, the file is restored to this location from bcakup.

As with all auxiliary set files, the file is deleted after successful TSPITR, or left for use in troubleshooting if TSPITR fails, regardless of whether it was an image copy created before TSPITR or restored by RMAN from backup during TSPITR.

TSPITR With CONFIGURE AUXNAME and Image Copies: Scenario

You have enough disk space to save image copies of your entire database for use in TSPITR. In preparation for the possibility that you need perform TSPITR, you perform the following tasks:

- Configure an AUXNAME for each datafile in your database using:

  ```
  CONFIGURE AUXNAME FOR DATAFILE n TO auxname_n;
  ```

- Every week on Sunday night you take an image copy of the database which is placed in the files referenced by the configured AUXNAMEs:

  ```
  BACKUP AS COPY DATAFILE n FORMAT auxname_n
  ```

  Note that if the image copies are all in the same location on disk and named similarly to the original datafiles, it is possible to use FORMAT or DB_FILE_NAME_CONVERT options of the BACKUP command and use BACKUP AS COPY DATABASE instead of performing individual backups of every datafile. For example if the configured auxnames are a simple translation of the location 'maindisk' to 'auxdisk', you could use the following backup command:

  ```
  BACKUP AS COPY DATABASE DB_FILE_NAME_CONVERT=(maindisk, auxdisk);
  ```

You are then prepared for TSPITR without restoring from backup. If, for example, an erroneous batch job started at November 15 2003, 19:00:00 updates incorrectly the tables in the tablespace PARTS. You could use the following command to perform TSPITR on tablespace PARTS:

```
RECOVER TABLESPACE parts UNTIL TIME 'November 15 2003, 19:00:00';
```

Because AUXNAMES are configured and refer to datafile copies from an SCN before the TSPITR target time, the auxiliary set and recovery set datafiles are not restored from backup. Instead the datafile copies are directly used in recovery, eliminating the restore overhead.

Note that at the end of the TSPITR, the tablespace PARTS will not be located in the original datafile locations, but in the auxname locations. If only the auxnames for the auxiliary set should be used (so that the recovery set is left in its original locations), then `CONFIGURE AUXNAME ... CLEAR` should be used before TSPITR is started. In such a case, though, note that the datafiles will have to be restored.

Customizing Initialization Parameters for the Automatic Auxiliary Instance

The automatic auxiliary instance looks for parameters in a file that is operating system dependent (for Unix this location is `?/rdbms/admin/params_auxint.ora`, where '?' stands for `ORACLE_HOME`, and the file is located on the node running the RMAN client, not necessarily the same node as the one running the database instances) This default parameter file for the automatic auxiliary instance is always searched when TSPITR is performed. If the file is not found RMAN does not generate an error.

Another way to specify parameters for the automatic auxiliary instance is to place the initializations parameter in a file, and then provide the location of these file with the `SET AUXILIARY INSTANCE PARAMETER` command before executing TSPITR. (Note that the path specified when using `SET AUXILIARY INSTANCE PARAMETER` is a path on the system running the RMAN client, not the target or auxiliary instances.)

RMAN defines the following basic parameters for the automatic auxiliary instance:

- `DB_NAME` - Same as db_name of the target database
- `DB_UNIQUE_NAME` - Generated, based on the `DB_NAME`, to be unique
- `DB_BLOCK_SIZE` - Same as the `DB_BLOCK_SIZE` of the target database
- `COMPATIBLE` - Same as the compatible setting of the target database

If `AUXILIARY DESTINATION` is used, RMAN also defines:

- `DB_CREATE_FILE_DEST` - Set to the auxiliary destination
- `CONTROL_FILES` - Generated filename in the auxiliary destination

When an auxiliary destination is specified, RMAN uses these two parameters in creating the auxiliary instance online logs and control files in the auxiliary destination.

If AUXILIARY DESTINATION is not used, then you must use LOG_FILE_NAME_CONVERT in an auxiliary instance parameter file to specify the online log file names. Otherwise, TSPITR fails when attempting to create the online logs for the automatic instance.

If AUXILIARY DESTINATION is not used and you do not use CONTROL_FILES in an auxiliary instance parameter file, the auxiliary instance will create one controlfile with an operating system-dependent name in an operating system dependent location. (In Unix, it defaults to ?/dbs/cntrl_@.dbf, where '?' stands for ORACLE_HOME and '@' stands for ORACLE_SID. For an automatic auxiliary instance, ORACLE_SID is randomly generated by RMAN).

It is rarely necessary, however, to alter the parameter file, especially if you provide an AUXILIARY DESTINATION argument to RECOVER TABLESPACE. If one of the six basic initialization parameters is overridden in the auxiliary instance parameter file, it might cause TSPITR to fail. However, other parameters besides these basic parameters can be added if needed. For example you can use DB_FILE_NAME_CONVERT to specify the names of the datafiles in the auxiliary set.

Performing RMAN TSPITR Using Your Own Auxiliary Instance

Oracle Corporation recommends that you allow RMAN to manage the creation and destruction of the auxiliary instance used during RMAN TSPITR. However, creating and using your own auxiliary instance is also supported. One reason you might want to do this is to exercise control of channels used in TSPITR. RMAN's automatic auxiliary instance uses the configured channels of the target database as the basis for the channels to configure on the auxiliary instance and use during backup. If you need different channel settings, and you do not want to use CONFIGURE to change the settings on the target database, you can operate your own auxiliary instance.

Preparing Your Own Auxiliary Instance for RMAN TSPITR

Creating an Oracle instance suitable for use as an auxiliary instance requires that you carry out all of the following steps:

- Step 1: Create an Oracle Password File for the Auxiliary Instance
- Step 2: Create an Initialization Parameter File for the Auxiliary Instance
- Step 3: Check Oracle Net Connectivity to the Auxiliary Instance
- Step 1: Start the Auxiliary Instance in NOMOUNT Mode

Step 1: Create an Oracle Password File for the Auxiliary Instance

For instructions on how to create and maintain Oracle password files, refer to the *Oracle Database Administrator's Guide.*

Step 2: Create an Initialization Parameter File for the Auxiliary Instance

Create a client-side initialization parameter file for the auxiliary instance on the machine where you will be running SQL*Plus to contorl the auxiliary instance. For this example, we will assume your parameter file is placed at `/tmp/initAux.ora`. Set the parameters described in the following table, making sure that paths in parameters like `DB_FILE_NAME_CONVERT`, `LOG_FILE_NAME_CONVERT` and `CONTROL_FILES` are all server-side paths, not client-side.

Table 10–2 Initialization Parameters in the Auxiliary Instance

Parameter	Mandatory?	Value
`DB_NAME`	YES	The same name as the target database.
`DB_UNIQUE_NAME`	YES	A value different from any database in the same Oracle home. For simplicity, specify *_dbname*. For example, if the target database name is `trgt`, then specify `_trgt`.
`LOG_FILE_NAME_ CONVERT`	YES	Patterns to generate filenames for the online redo logs of the auxiliary database based on the online redo log names of the target database. Query `V$LOGFILE.MEMBER`, to obtain target instance online log names, and ensure that the conversion pattern matches the format of the filename displayed in the view. This parameter is the only way to name the online redo logs for the auxiliary instance. Without it, TSPITR will fail when trying to open the auxiliary instance because the online logs cannot be created. **Note:** Some platforms do not support ending patterns in a forward or backward slash (\ or /).
`REMOTE_LOGIN_ PASSWORDFILE`	YES	Set to `EXCLUSIVE` when connecting to the auxiliary instance by means of a password file. Otherwise, set to `NONE`.
`COMPATIBLE`	YES	The same value as the parameter in the target database.

Table 10–2 Initialization Parameters in the Auxiliary Instance

Parameter	Mandatory?	Value
DB_BLOCK_SIZE	YES	If this initialization parameter is set in the target database, then it must be set to the same value in the auxiliary instance.
DB_FILE_NAME_CONVERT	NO	Patterns to convert filenames for the datafiles of the auxiliary database. You can use this parameter to generate filenames for those files that you did not name with SET NEWNAME or CONFIGURE AUXNAME. Obtain the datafile filenames by querying V$DATAFILE.NAME, and ensure that the conversion pattern matches the format of the filename displayed in the view. You can also specify this parameter on the RECOVER command itself. **Note:** Some platforms do not support ending patterns in a forward or backward slash (\ or /). **See Also:** "Using DB_FILE_NAME_CONVERT to Name Auxiliary Set Datafiles" on page 10-16
CONTROL_FILES	NO	Filenames that do not conflict with the control file names of the target instance (or any other existing file).

Set other parameters as needed, including the parameters that allow you to connect as SYSDBA through Oracle Net.

Following are examples of the initialization parameter settings for the auxiliary instance:

```
DB_NAME=trgt
DB_UNIQUE_NAME=_trgt
CONTROL_FILES=/tmp/control01.ctl
DB_FILE_NAME_CONVERT=('/oracle/oradata/trgt/','/tmp/')
LOG_FILE_NAME_CONVERT=('/oracle/oradata/trgt/redo','/tmp/redo')
REMOTE_LOGIN_PASSWORDFILE=exclusive
COMPATIBLE =10.1.0
DB_BLOCK_SIZE=8192
```

Note: After setting these initialization parameters, ensure that you do not overwrite the initialization settings for the production files at the target database.

See Also: *Oracle Net Services Administrator's Guide* for more information about Oracle Net

Step 3: Check Oracle Net Connectivity to the Auxiliary Instance

The auxiliary instance must have a valid net service name. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance.

Preparing RMAN Commands for TSPITR with Your Own Auxiliary Instance

If you are running your own auxiliary instance, then you may find that the sequence of commands required for TSPITR is quite long, if you allocate a complex channel configuration for restoring from backup, or if you are not using `DB_FILE_NAME_CONVERT` to control file naming.

You may wish to store the sequence of commands for TSPITR in a command file, a text file under the host operating system. This command file can be read into RMAN using the `@` command (or the `CMDFILE` command line argument when starting RMAN) to execute the series of commands in the command file.

See "Using RMAN with Command Files" on page 1-4 for more details.

Planning Channels for TSPITR with Your Own Auxiliary Instance

When you run your own auxiliary instance, the default behavior is to use the automatic channel configuration of the target instance. However, if you decide to allocate your own channel configuration, you can do so by including the `ALLOCATE AUXILIARY CHANNEL` commands in a RUN block along with the `RECOVER TABLESPACE` command for TSPITR. Plan out these commands, if necessary, and add them to the sequence of commands you will run to perform your TSPITR.

See the example in "Executing TSPITR With Your Own Auxiliary Instance: Scenario" on page 10-27 for details of how to include channel allocation in your TSPITR script.

Planning Datafile Names with Your Own Auxiliary Instance: SET NEWNAME

You may wish to use `SET NEWNAME` commands, either to refer to existing image copies of auxiliary set files to improve TSPITR performance, or to assign new names to the recovery set files for after TSPITR. Plan out these commands, if necessary, and add them to the sequence of commands you will run to perform your TSPITR.

Executing TSPITR with Your Own Auxiliary Instance

With the preparations complete and your TSPITR commands completely planned, you are now ready to carry out your TSPITR. The following steps are required:

- Step 1: Start the Auxiliary Instance in NOMOUNT Mode
- Step 2: Connect the RMAN Client to Target and Auxiliary Instances
- Step 3: Execute the RECOVER TABLESPACE Command

Step 1: Start the Auxiliary Instance in NOMOUNT Mode

Before beginning RMAN TSPITR, use SQL*Plus to connect to the auxiliary instance and start it in NOMOUNT mode, specifying a parameter file if necessary. For example:

```
SQL> CONNECT SYS/oracle@aux AS SYSDBA
SQL> STARTUP NOMOUNT PFILE='/tmp/initAux.ora'
```

Remember that the path for the PFILE will be a client-side path, on the machine from which you run SQL*Plus, not a server-side path.

Because the auxiliary instance does not yet have a control file, you can only start the instance in NOMOUNT mode. Do not create a control file or try to mount or open the auxiliary instance for TSPITR.

Step 2: Connect the RMAN Client to Target and Auxiliary Instances

Start RMAN connecting to the target and the manually created auxiliary instance:

```
% rman target / auxiliary sysuser/syspwd@auxiliary_service_name
```

Step 3: Execute the RECOVER TABLESPACE Command

Now you are ready to run your TSPITR commands. In the simplest case, just execute the RECOVER TABLESPACE... UNTIL command at the RMAN prompt:

```
RMAN> RECOVER TABLESPACE ts1, ts2... UNTIL TIME 'time'
```

If you want to use ALLOCATE CHANNEL or SET NEWNAME then create a RUN block which includes those commands before the RECOVER TABLESPACE command.

```
RUN {
   ALLOCATE CHANNEL c1 DEVICE TYPE DISK;
   ALLOCATE CHANNEL c2 DEVICE TYPE SBT;
  # and so on...
   RECOVER TABLESPACE ts1, ts2 UNTIL TIME 'time';
```

```
}
```

Using a Command File for TSPITR Entering a lengthy series of commands in a RUN block can be error-prone. To avoid making mistakes entering the sequence of commands, create a command file (called, for example, `/tmp/tspitr.rman`) to store the whole sequence of commands for your TSPITR. Review it carefully to catch any errors. Then run the command file at the RMAN prompt, using this command:

```
RMAN> @/tmp/tspitr.rman ;
```

The results will be the same as in the previous example.

Executing TSPITR With Your Own Auxiliary Instance: Scenario

The following example shows the execution of a `RECOVER TABLESPACE... UNTIL` operation using the following features of RMAN TSPITR:

- Managing your own auxiliary instance
- Configuring channels for restore of backups from disk and sbt
- Using recoverable image copies for some auxiliary set datafiles using `SET NEWNAME`
- Specifying new names for recovery set datafiles using `SET NEWNAME`

The process used is as follows:

1. Prepare the auxiliary instance as described in "Preparing Your Own Auxiliary Instance for RMAN TSPITR" on page 10-22. Specify "`tspitr`" as the password for the auxiliary instance in the password file, and set up the auxiliary instance parameter file `/bigtmp/init_tspitr_prod.ora` with the following settings:

   ```
   db_name=PROD
   db_unique_name=tspitr_PROD
   control_files=/bigtmp/tspitr_cntrl.f'
   db_file_name_convert=('?/oradata/prod', '/bigtmp')
   log_file_name_convert=('?/oradata/prod', '/bigtmp')
   compatible=10.1.0
   block_size=8192
   remote_login_password=exclusive
   ```

2. Create service name `pitprod` for the auxiliary instance, and check for connectivity.

3. Start the auxiliary instance in NOMOUNT state, as shown:

```
$ sqlplus
SQL> connect sys/tspitr@pit_prod as sysdba
SQL> startup nomount pfile=/bigtmp/init_tspitr_prod.ora
```

4. Start up RMAN, connecting to the auxiliary instance:

```
% rman target / auxiliary sys/tspitr@pit_prod
```

5. Enter the following commands, in a RUN block, to set up and execute the TSPITR:

```
run {
# Specify NEWNAMES for recovery set datafiles
    SET NEWNAME FOR DATAFILE '?/oradata/prod/clients01.f'
                                TO '?/oradata/prod/clients01_rec.f';
            SET NEWNAME FOR DATAFILE '?/oradata/prod/clients02.f'
                                TO '?/oradata/prod/clients02_rec.f';
            SET NEWNAME FOR DATAFILE '?/oradata/prod/clients03.f'
                                TO '?/oradata/prod/clients03_rec.f';
            SET NEWNAME FOR DATAFILE '?/oradata/prod/clients04.f'
                                TO '?/oradata/prod/clients04_rec.f';

# Specified newnames for some of the auxiliary set
# datafiles that have a valid image copy to avoid restores:
            SET NEWNAME FOR DATAFILE '?/oradata/prod/system01.f'
                                TO '/backups/prod/system01_monday_noon.f';
            SET NEWNAME FOR DATAFILE '?/oradata/prod/system02.f'
                                TO '/backups/prod/system02_monday_noon.f';
            SET NEWNAME FOR DATAFILE '?/oradata/prod/undo01.f'
                                TO '/backups/prod/undo01_monday_noon.f';

# Specified the disk and SBT channels to use
            allocate auxiliary channel c1 device type disk;
            allocate auxiliary channel c2 device type disk;
            allocate auxiliary channel t1 device type sbt;
            allocate auxiliary channel t2 device type sbt;

# Recovered the clients tablespace to 24 hours ago:
    RECOVER TABLESPACE clients UNTIL TIME 'sysdate-1';
   }
```

If the TSPITR operation is successful, then the results are:

- The recovery set datafiles are registered in the target database control file under the names specified with SET NEWNAME, with their contents as of the time specified time for the TSPITR.
- The auxiliary files are removed by RMAN, including the control files, online logs and auxiliary set datafiles of the auxiliary instance
- The auxiliary instance is shut down

If the TSPITR operation fails, the auxiliary files are left on disk for troubleshooting purposes. If RMAN created the auxiliary instance, it is shut down; otherwise it is left in whatever state it was in when the TSPITR operation failed.

Troubleshooting RMAN TSPITR

A variety of problems can cause TSPITR to fail before the process is complete.

- There can be name conflicts between files already in the target database, filenames assigned by the `SET NEWNAME` or `CONFIGURE AUXNAME` commands, and filenames generated by the effect of the `DB_FILE_NAME_CONVERT` parameter.
- When RMAN exports the metadata about recovered objects from the auxiliary instance, it uses space in the temporary tablespace for sorting. If there is insufficient space in the temporary tablespace for the sorting operation, you need to increase the amount of sort space available.

Troubleshooting TSPITR Example: Filename Conflicts

If your uses of `SET NEWNAME`, `CONFIGURE AUXNAME` and `DB_FILE_NAME_CONVERT` cause multiple files in the auxiliary or recovery sets to have the same name, RMAN will report an error during TSPITR. To correct the problem, use different values for these parameters to eliminate the duplicate name.

Troubleshooting TSPITR Example: Insufficient Sort Space during Export

In this case, you need to edit the `recover.bsq` file, wherever it resides on your host platform. For instance, on UNIX, it is located in `$ORACLE_HOME/rdbms/admin`. This file contains the following:

```
#
# tspitr_7: do the incomplete recovery and resetlogs.  This member is used once.
#
```

```
define tspitr_7
<<<
# make the control file point at the restored datafiles, then recover them
RECOVER CLONE DATABASE TABLESPACE &1&;
ALTER CLONE DATABASE OPEN RESETLOGS;
# PLUG HERE the creation of a temporary tablespace if export fails due to lack
# of temporary space.
# For example in Unix these two lines would do that:
# sql clone "create tablespace aux_tspitr_tmp
#           datafile ''/tmp/aux_tspitr_tmp.dbf'' size 500K";
}
>>>
```

Remove the '#' symbols from the last two lines of comments and modify the statement to create a temporary tablespace. Retry the TSPITR operation, increasing the size of the tablespace until the export operation succeeds.

Troubleshooting: Restarting Manual Auxiliary Instance After TSPITR Failure

If you are managing your own auxiliary instance and there is a failure in TSPITR, then before you can try TSPITR again, you must shut down the auxiliary instance, correct the problem which interfered with TSPITR, and then bring the auxiliary instance back to NOMOUNT before trying TSPITR again.

11

Duplicating a Database with Recovery Manager

This chapter describes how to use the DUPLICATE command to create a duplicate database for testing purposes. This chapter contains these topics:

- Creating a Duplicate Database: Overview
- Generating Files for the Duplicate Database
- Preparing the Auxiliary Instance for Duplication: Basic Steps
- Creating a Duplicate Database on a Local or Remote Host
- Database Duplication Examples

> **See Also:** *Oracle Data Guard Concepts and Administration* to learn how to create a standby database with the DUPLICATE command

Creating a Duplicate Database: Overview

You can use the RMAN `DUPLICATE` command to create a **duplicate database** from target database backups while still retaining the original target database. The duplicate database can be identical to the original database or contain only a subset of the original tablespaces.

A duplicate database is a copy of a target database that you can run independently for a variety of purposes. For example, you can use it to:

- Test backup and recovery procedures
- Export data such as a table that was inadvertently dropped from the production database, and then import it back into the production database

For example, you can duplicate the production database on `host1` to `host2`, and then use the duplicate database on `host2` to practice restore and recovery scenarios while the production database on `host1` continues as usual.

A duplicate database is distinct from a standby database, although both types of databases are created with the `DUPLICATE` command. A standby database is a copy of the primary database that you can update continually or periodically with archived logs from the primary database. If the primary database is damaged or destroyed, then you can perform failover to the standby database and effectively transform it into the new primary database. A duplicate database, on the other hand, cannot be used in this way: it is not intended for failover scenarios and does *not* support the various standby recovery and failover options.

> **See Also:** *Oracle Data Guard Concepts and Administration* to learn how to create a standby database with the `DUPLICATE` command

How Recovery Manager Duplicates a Database

To prepare for database duplication, you must first create an **auxiliary instance** as described in "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9. For the duplication to work, you must connect RMAN to both the target (primary) database and an auxiliary instance started in `NOMOUNT` mode.

You must have at least one auxiliary channel allocated on the auxiliary instance. The principal work of the duplication is performed by the auxiliary channel, which starts a server session on the duplicate host. This channel then restores the necessary backups of the primary database, uses them to create the duplicate database, and initiates recovery.

So long as RMAN is able to connect to the primary and auxiliary instances, the RMAN client can run on any machine. However, all backups and archived logs used for creating and recovering the duplicate database must be accessible by the server session on the duplicate host. If the duplicate host is not the same as the primary host, then you must make backups on disk on the primary host available to the duplicate host with the same full path name as in the primary database. When using disk backups, you can accomplish this goal in any of the following ways:

- Manually transfer the backups from the primary host to the remote host to an identical path.
- Manually transfer the backups from the primary host to the remote host to a new path, and then run the `CATALOG` command to add these copies to the RMAN repository.
- Use NFS or shared disks and make sure that the same path is accessible in the remote host.

When using tape backups, you must make the tapes containing the backups accessible to the remote node, either by physically moving the tape to the remote host or by means of a network tape server.

As part of the duplicating operation, RMAN automates the following steps:

- Creates a control file for the duplicate database
- Restores the target datafiles to the duplicate database and performs incomplete recovery by using all available incremental backups and archived logs
- Shuts down and starts the auxiliary instance (refer to "Task 4: Start the Auxiliary Instance" on page 11-11 for issues relating to client-side versus server-side initialization parameter files)
- Opens the duplicate database with the `RESETLOGS` option after incomplete recovery to create the online redo logs (except when running `DUPLICATE ... FOR STANDBY`, in which case RMAN does not open the database)
- Generates a new, unique DBID for the duplicate database (except when you create a standby database with `DUPLICATE ... FOR STANDBY`, in which case RMAN does *not* create a unique DBID)

During duplication, RMAN *must* perform incomplete recovery because the online redo logs in the target are not backed up and cannot be applied to the duplicate database. The farthest that RMAN can go in recovery of the duplicate database is the most recent redo log archived by the target database.

> **See Also:** *Oracle Data Guard Concepts and Administration* to learn how to create a standby database with RMAN

Database Duplication Options

When duplicating a database, you have the following options:

- You can run the DUPLICATE command with or without a recovery catalog
- You can skip read-only tablespaces with the SKIP READONLY clause. Read-only tablespaces are included by default. If you omit them, then you can add them later.
- You can exclude tablespaces from the duplicate database with the SKIP TABLESPACE clause. You can exclude any tablespace except the SYSTEM tablespace or tablespaces containing rollback or undo segments.
- You can create the duplicate database in a new host. If the directory structure is the same on the new host, then you can specify the NOFILENAMECHECK option and reuse the target datafile filenames for the duplicate datafiles.
- You can duplicate a target database on a traditional file system to an ASM or Oracle Managed Files location.
- You can recover the duplicate database to a past point in time, using use the SET UNTIL command or DUPLICATE ... UNTIL command. By default, the DUPLICATE command creates the duplicate database by using the most recent backups of the target database and then performs recovery to the most recent consistent point contained in the incremental backups and archived logs.
- You can register the duplicate database in the same recovery catalog as the target database. This option is possible because RMAN gives the duplicate database a new, unique DBID during duplication.

  > **Note:** If you copy the target database by means of operating system utilities, then the DBID of the copied database remains the same as the original database. To register the copy database in the same recovery catalog with the original, you must change the DBID with the DBNEWID utility (refer to *Oracle Database Utilities*).

- In some cases, you can set the duplicate database DB_NAME differently from the target database DB_NAME. More specifically, if the duplicate database exists in the same Oracle home as the target, then the DB_NAME initialization parameter

must be different. If the duplicate database is in a different Oracle home from the target database, then the DB_NAME setting for the duplicate database must be unique among databases in its Oracle home. This is true whether or not the duplicate database is on the same host as the target.

Duplicating a Database: Prerequisites and Restrictions

RMAN duplication involves a number of prerequisites, restrictions, and caveats. Review the restrictions section of the DUPLICATE command in the *Oracle Database Recovery Manager Reference* for a complete list.

Generating Files for the Duplicate Database

When duplicating a database, RMAN creates the required database files. This section describes these stages of file creation:

- Creating the Duplicate Control Files
- Creating the Duplicate Online Redo Logs
- Renaming Datafiles When Duplicating a Database

Creating the Duplicate Control Files

The DUPLICATE command creates the control files by using the names listed in the initialization parameter file of the auxiliary instance. When choosing names for the duplicate database control files, make sure that you set the initialization parameter settings correctly so that you do not overwrite the production files at the target database.

Creating the Duplicate Online Redo Logs

Table 11–1 lists the options for creating the names of the duplicate online redo logs. The options appear in the order of precedence.

Table 11–1 Order of Precedence for Online Redo Log Filename Creation

Order	Method	Result
1	Specify the LOGFILE clause of DUPLICATE command.	Creates online redo logs as specified.

Table 11–1 Order of Precedence for Online Redo Log Filename Creation

Order	Method	Result
2	Set LOG_FILE_NAME_CONVERT initialization parameter.	Transforms target filenames, for example, from log_* to duplog_*. Note that you can specify multiple conversion pairs. This parameter allows the redo log to exist as long as the size matches, because it uses the REUSE parameter when creating the logs.
3	Do none of the preceding steps.	Makes the duplicate filenames the same as the target filenames. You must specify the NOFILENAMECHECK option when using this method and the duplicate database should be in a different host.

The order of precedence determines how RMAN renames the online redo logs. For example, if you specify both the LOGFILE clause and the LOG_FILE_NAME_CONVERT parameter, then RMAN uses the LOGFILE clause. If you specify neither of the first two options, then RMAN uses the original target redo log filenames for the duplicate database files.

Caution: If the target and duplicate databases are in the same host, then do not use the name of an online redo log currently in use by the target database. Also, do not use the name of an online log currently in use by the target database if the duplicate database is in a different host and NOFILENAMECHECK is not used.

Renaming Datafiles When Duplicating a Database

There are several methods of specifying new names to be used for the datafiles of your duplicate database. Listed in order of precedence, they are:

- Use the SET NEWNAME command, within a RUN block enclosing both the SET NEWNAME commands and the DUPLICATE command.
- Use the CONFIGURE AUXNAME command to specify new names for existing datafiles, before using the DUPLICATE command.
- Specify the DB_FILE_NAME_CONVERT parameter to the DUPLICATE command, to specify a rule for converting filenames for any datafiles not renamed using SET NEWNAME or CONFIGURE AUXNAME. Note that you can specify multiple conversion pairs, and use ASM disk groups.

- Set the DB_FILE_NAME_CONVERT initialization parameter, subject to the same semantics and limitations as the DB_FILE_NAME_CONVERT parameter to the DUPLICATE command.

If yo do not use any of these options, then the duplicate database will reuse the original datafile filenames from the target database.

Preventing Filename Checking

It is possible for CONFIGURE AUXNAME, SET NEWNAME, or DB_FILE_NAME_CONVERT to generate a name that is already in use in the target database. In this case, specify NOFILENAMECHECK to avoid an error message. For example, assume that the host A database has two files: datafile 1 is named /oracle/data/file1.f and datafile 2 is named /oracle/data/file2.f. When duplicating to host B, you use a configured channel to duplicate as follows:

```
RUN
{
  SET NEWNAME FOR DATAFILE 1 TO /oracle/data/file2.f; # rename df 1 as file2.f
  SET NEWNAME FOR DATAFILE 2 TO /oracle/data/file1.f; # rename df 2 as file1.f
  DUPLICATE TARGET DATABASE TO newdb;
}
```

Even though you issued SET NEWNAME commands for all the datafiles, the DUPLICATE command fails because the duplicate filenames are still in use in the target database. Although datafile 1 in the target is not using /oracle/data/file2.f, and datafile 2 in the target is not using /oracle/data/file1.f, the target filename is used by one of the duplicate datafiles and so you must specify NOFILENAMECHECK to avoid an error.

Skipping Read-Only Tablespaces When Duplicating a Database

When you specify SKIP READONLY, RMAN does not duplicate the datafiles of read-only tablespaces. After duplication is complete, you can query the views in the duplicate database described in Table 11–2 and Table 11–3 to determine which datafiles were skipped. The STATUS and ENABLED columns are the key to determining the current status of the duplicate datafile.

Table 11–2 Read-Only Tablespaces in V$DATAFILE View on Duplicate Database

In the column ...	The value is ...
STATUS	OFFLINE
ENABLED	READ ONLY

Table 11–2 Read-Only Tablespaces in V$DATAFILE View on Duplicate Database

In the column ...	The value is ...
NAME	MISSINGxxx

Table 11–3 Read-Only Tablespaces in Data Dictionary Views on Duplicate Database

View	In the column ...	The value is ...
DBA_DATA_FILES	STATUS	AVAILABLE
DBA_TABLESPACES	STATUS	READ ONLY

Skipping OFFLINE NORMAL Tablespaces When Duplicating a Database

When tablespaces are taken offline with the OFFLINE NORMAL option before a DUPLICATE operation, RMAN does not duplicate the datafiles of these tablespaces. After duplication, you can manually add or drop these tablespaces. Query the views in the duplicate database described in Table 11–4 and Table 11–5 to determine which datafiles are offline, based on the STATUS and ENABLED columns.

Table 11–4 Offline Tablespaces in V$ Views on Duplicate Database

In the column ...	The value is ...
STATUS	OFFLINE
ENABLED	DISABLED
NAME	MISSINGxxx

Table 11–5 Offline Tablespaces in Data Dictionary Views on Duplicate Database

View	In the column ...	The value is ...
DBA_DATA_FILES	STATUS	AVAILABLE
DBA_TABLESPACES	STATUS	OFFLINE

When you take a tablespace offline with the IMMEDIATE option, RMAN duplicates rather than skips the tablespace because this tablespace requires recovery. As with online tablespaces, RMAN requires a valid backup for duplication.

Preparing the Auxiliary Instance for Duplication: Basic Steps

Perform these tasks before performing RMAN duplication:

- Task 1: Create an Oracle Password File for the Auxiliary Instance
- Task 2: Ensure Oracle Net Connectivity to the Auxiliary Instance
- Task 3: Create an Initialization Parameter File for the Auxiliary Instance
- Task 4: Start the Auxiliary Instance
- Task 5: Mount or Open the Target Database
- Task 6: Make Sure You Have the Necessary Backups and Archived Redo Logs
- Task 7: Allocate Auxiliary Channels if Automatic Channels Are Not Configured

Task 1: Create an Oracle Password File for the Auxiliary Instance

For instructions on how to create and maintain Oracle password files, refer to *Oracle Database Administrator's Guide*.

Task 2: Ensure Oracle Net Connectivity to the Auxiliary Instance

The auxiliary instance must be accessible through Oracle Net. Before proceeding, start SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.

Task 3: Create an Initialization Parameter File for the Auxiliary Instance

Create a client-side initialization parameter file for the auxiliary instance, and set at least the parameters described in the following table.

Parameter	You must specify:
DB_NAME	The same name used in the DUPLICATE command. You must set the DB_NAME parameter in the duplicate parameter file to the same database name specified in the DUPLICATE command. You cannot use the same database name for the target and duplicate when the duplicate is in the same Oracle home as the target. If the duplicate is in a different Oracle home from the target, then its DB_NAME just has to differ from other database names in the same Oracle home.

Parameter	You must specify:
CONTROL_FILES	Refer to "Creating the Duplicate Control Files" on page 11-5.

You can also set the initialization parameters described in the following table.

Initialization Parameter	You must specify:
DB_FILE_NAME_CONVERT	Refer to "Renaming Datafiles When Duplicating a Database" on page 11-6. You can also specify this parameter on the DUPLICATE command itself.
LOG_FILE_NAME_CONVERT	Refer to "Creating the Duplicate Online Redo Logs" on page 11-5.

Set other initialization parameters, including the parameters that allow you to connect as SYSDBA through Oracle Net, as needed. When duplicating to the same host or to a new host with a different file system, pay attention to all initialization parameters specifying path names. Verify that all paths are accessible on the host where the database is being duplicated.

Following are examples of the initialization parameter settings for the duplicate database:

```
DB_NAME=newdb
CONTROL_FILES=(/dup/oracle/oradata/trgt/control01.ctl,
               /dup/oracle/oradata/trgt/control02.ctl)
# note that the following two initialization parameters have equivalents
# on the DUPLICATE command itself
DB_FILE_NAME_CONVERT=(/oracle/oradata/trgt/,/dup/oracle/oradata/trgt/)
LOG_FILE_NAME_CONVERT=(/oracle/oradata/trgt/redo,/dup/oracle/oradata/trgt/redo)
```

After you create the client-side initialization parameter file, you can run the CREATE SPFILE command from SQL*Plus to create a server-side initialization parameter file. You can run this command before or after instance startup. For example, you can create a server-side parameter file in the default location as follows, specifying the filename of the client-side initialization parameter file in the FROM clause:

```
CREATE SPFILE FROM PFILE='/tmp/initDUPDB.ora';
```

A server-side parameter file in the default location is an advantage when duplicating a database because you do not need to specify the PFILE parameter on the DUPLICATE command. Because RMAN shuts down and restarts the auxiliary

instance as part of the duplication process, you must tell RMAN which client-side file to use if you use a client-side parameter file. It is highly recommended that you create a server-side parameter file for use in database duplication.

Task 4: Start the Auxiliary Instance

Before beginning RMAN duplication, use SQL*Plus to connect to the auxiliary instance and start it in NOMOUNT mode (specifying a client-side parameter file if necessary). In this example, oracle is the password for the user with SYSDBA authority and aux is the net service name for the auxiliary instance:

```
CONNECT SYS/oracle@aux AS SYSDBA
-- start instance with the server parameter file
STARTUP FORCE NOMOUNT
```

Because the auxiliary instance does not yet have a control file, you can only start the instance in NOMOUNT mode. Do not create a control file or try to mount or open the auxiliary instance.

RMAN shuts down and restarts the auxiliary instance as part of the duplication. Hence, it is a good idea to create a server-side initialization parameter file for the auxiliary instance in the default location.

If you do *not* have a server-side initialization parameter file for the auxiliary instance in the default location, then you must specify the client-side initialization parameter file with the PFILE parameter on the DUPLICATE command. The client-side parameter file for the auxiliary instance must reside on the same host as the RMAN client used to perform the duplication.

Task 5: Mount or Open the Target Database

Before beginning RMAN duplication, connect SQL*Plus to the target database and mount or open it if it is not already mounted or open. For example, enter:

```
-- connect to target database
CONNECT SYS/oracle@trgt
-- mount or open target database
STARTUP
```

If necessary, you can use a client-side initialization file to start up the target instance.

Task 6: Make Sure You Have the Necessary Backups and Archived Redo Logs

Make sure backups all target datafiles are accessible on the duplicate host. If you do not have backups of everything, then the duplicate operation fails. The database backup does not have to be a whole database backup: you can use a mix of full and incremental backups of individual datafiles.

Archived redo logs required to recover the duplicate database to the desired point in time must be accessible by the node where the duplicate database is to be created, either as backups (for instance, on a media manager) or as image copies (or the actual archived redo logs), either copied to the local disk of the node that contains the duplicate database, or possibly mounted across a network by some means such as NFS.

Task 7: Allocate Auxiliary Channels if Automatic Channels Are Not Configured

Start RMAN with a connection to the target database, the auxiliary instance, and, if applicable, the recovery catalog database. You can start the RMAN client on any host so long as it can connect to all the instances. If the auxiliary instance requires a client-side initialization parameter file, then this file must exist on the same host that runs the RMAN client.

In this example, a connection is established to three instances, all through the use of net service names:

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb AUXILIARY SYS/oracle@aux
```

If you do not have automatic channels configured, then before issuing the `DUPLICATE` command, manually allocate at least one auxiliary channel within the same `RUN` command. The channel type (`DISK` or `sbt`) must match the media where the backups of the target database are located. If the backups reside on disk, then the more channels you allocate, the faster the duplication will be. For tape backups, limit the number of channels to the number of devices available.

```
RUN
{
  # to manually allocate a channel of type sbt issue:
  ALLOCATE AUXILIARY CHANNEL ch1 DEVICE TYPE sbt;

  # to manually allocate three auxiliary channels for disk issue (specifying
  # whatever channel id that you want):
  ALLOCATE AUXILIARY CHANNEL aux1 DEVICE TYPE DISK;
  ALLOCATE AUXILIARY CHANNEL aux2 DEVICE TYPE DISK;
  ALLOCATE AUXILIARY CHANNEL aux3 DEVICE TYPE DISK;
  .
```

```
  .
  .
  DUPLICATE ...
}
```

Note: If you configure automatic channels, then RMAN can use configured channels for duplication even if they do not specify the AUXILIARY option. Nevertheless, if the auxiliary channels need some special parameters (for example, to point to a different media management subsystem), then you can configure an automatic channel with the AUXILIARY option.

Creating a Duplicate Database on a Local or Remote Host

The proceture to create a duplicate database depends on your configuration.

Duplicating a Database on a Remote Host with the Same Directory Structure

The simplest case is to duplicate the database to a different host and to use the same directory structure. In this case, you do not need to change the initialization parameter file or set new filenames for the duplicate datafiles.

1. Follow the steps in "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9.
2. Run the DUPLICATE command, making sure to do the following:
   - If automatic channels are not configured, then allocate at least one auxiliary channel.
   - Specify the NOFILENAMECHECK parameter on the DUPLICATE command.
   - Specify the PFILE parameter if starting the auxiliary instance with a client-side parameter file. The client-side parameter file must exist on the same host as the RMAN client used to perform the duplication.

   The following example assumes that the RMAN client is running on the duplicate host. It duplicates the database with an automatic channel, specifies a client-side initialization parameter file, and specifies the NOFILENAMECHECK option:

   ```
   DUPLICATE TARGET DATABASE TO dupdb
     # specify client-side parameter file (on same host as RMAN client) for
     # auxiliary instance if necessary
   ```

```
    PFILE = /dup/oracle/dbs/initDUPDB.ora
    NOFILENAMECHECK;
```

RMAN automatically allocates the configured channels, then uses all incremental backups, archived redo log backups, and archived redo logs to perform incomplete recovery. Finally, RMAN opens the database with the RESETLOGS option to create the online redo logs.

Duplicating a Database on a Remote Host with a Different Directory Structure

If you create the duplicate database on a host with a different directory structure, then you must change several initialization parameters, in order to generate new filenames for the duplicate database datafiles on the new directory structure. The different methods available for doing so are described in "Renaming Datafiles When Duplicating a Database" on page 11-6.

Converting Filenames with Only Initialization Parameters

This procedure assumes that you use only initialization parameters to rename the duplicate datafiles and log files.

1. Follow the steps in "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9. You can either create a client-side parameter file, or copy the parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure using operating system utilities. Verify that you have done all of the following:
   - Set all initialization parameters that end in _DEST or _PATH and specify a path name.
   - Set DB_FILE_NAME_CONVERT so that it captures *all* the target datafiles and converts them appropriately, for example, from /oracle/oradata/ to /dup/oracle/oradata/.
   - Set LOG_FILE_NAME_CONVERT so that it captures *all* the online redo logs and converts them appropriately, for example, /oracle/oradata/redo to /dup/oracle/oradata/redo.

> **Note:** You can set multiple conversion pairs in DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT. For example, you can specify that DB_FILE_NAME_CONVERT changes /disk1/dbs to /dup1/dbs and /disk2/dbs to /dub2/dbs. Also, while this example focuses on using DB_FILE_NAME_CONVERT, remember that you can use the CONFIGURE AUXNAME or SET NEWNAME commands to rename individual datafiles if you cannot easily generate all of your desired filenames using DB_FILE_NAME_CONVERT commands.

2. Perform the following operations when running the duplication:

   - If automatic channels are not configured, then allocate at least one auxiliary channel.
   - If using a client-side parameter file to start the auxiliary instance, specify the PFILE parameter.

   The following example assumes that the duplicate host can access the same media manager as the primary database host. The example duplicates the database with an automatic sbt channel and uses a server-side parameter file located on the duplicate host to restart the auxiliary instance:

```
DUPLICATE
  TARGET DATABASE TO dupdb
  DEVICE TYPE sbt # restores from tape backups;
# DUPLICATE DEVICE TYPE sbt works only if the sbt device is configured
# by CONFIGURE CHANNEL, CONFIGURE DEVICE TYPE, or CONFIGURE DEFAULT DEVICE.
```

   RMAN uses all incremental backups, archived redo log backups, and archived redo logs to perform incomplete recovery. RMAN then shuts down, starts, and opens the database with the RESETLOGS option to create the online redo logs.

Converting Filenames with Only DUPLICATE Parameters

This procedure assumes that you use the DB_FILE_NAME_CONVERT parameter of the DUPLICATE command to rename the duplicate datafiles, and the LOGFILE clause to specify names and sizes for the online redo logs.

Perform the following operations when running the duplication:

- If automatic auxiliary channels are not configured, then allocate at least one auxiliary channel.

- Specify the names and sizes for the duplicate database redo logs in the LOGFILE clause.
- Specify new filenames for the duplicate database datafiles with the DB_FILE_NAME_CONVERT parameter.
- If using a client-side parameter file to start the auxiliary instance, specify the PFILE parameter.

The following example duplicates the database with an automatic channel and specifies an initialization parameter file:

```
DUPLICATE TARGET DATABASE TO dupdb
  # specify client-side parameter file for auxiliary instance if necessary
  PFILE = /dup/oracle/dbs/initDUPDB.ora
  DB_FILE_NAME_CONVERT=(/oracle/oradata/trgt/,/dup/oracle/oradata/trgt/)
  LOGFILE
    '/dup/oracle/oradata/trgt/redo01.log' SIZE 200K,
    '/dup/oracle/oradata/trgt/redo02.log' SIZE 200K,
    '/dup/oracle/oradata/trgt/redo03.log' SIZE 200K;
```

Converting Filenames with SET NEWNAME

This procedure assumes that you use the SET NEWNAME command to rename the duplicate datafiles.

1. Follow the steps in "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9, making sure to use an operating system utility to copy the parameter file from its location in the target host directory structure to the same location in the duplicate host. Set all initialization parameters that end in _DEST or _PATH and specify a path name.
2. Perform the following operations when running the duplication:
   - If automatic auxiliary channels are not configured, then allocate at least one auxiliary channel.
   - If desired, specify the same number of redo log members and groups that are used in the target database.
   - Specify new filenames for the duplicate database datafiles.
   - If you use a client-side parameter file to start the auxiliary instance, then specify the PFILE parameter.

   The following example uses automatic channels and a default server-side initialization parameter file for the database duplication, and uses the LOGFILE clause to specify names and sizes for the online redo logs:

```
RUN
{
  # set new filenames for the datafiles
  SET NEWNAME FOR DATAFILE 1 TO '/dup/oracle/oradata/trgt/system01.dbf';
  SET NEWNAME FOR DATAFILE 2 TO '/dup/oracle/oradata/trgt/undotbs01.dbf';
  . . .
  # issue the duplicate command
  DUPLICATE TARGET DATABASE TO dupdb
  # create at least two online redo log groups
  LOGFILE
    GROUP1
    (
      '/dup/oracle/oradata/trgt/redo01a.log',
      '/dup/oracle/oradata/trgt/redo01b.log',
      '/dup/oracle/oradata/trgt/redo01c.log';
    ) SIZE 200K,
    GROUP2
    (
      '/dup/oracle/oradata/trgt/redo02a.log',
      '/dup/oracle/oradata/trgt/redo02b.log',
      '/dup/oracle/oradata/trgt/redo02c.log';
    ) SIZE 200K,
    GROUP3
    (
      '/dup/oracle/oradata/trgt/redo03a.log',
      '/dup/oracle/oradata/trgt/redo03b.log',
      '/dup/oracle/oradata/trgt/redo03c.log';
    ) SIZE 200K;
```

RMAN uses all incremental backups, archived redo log backups, and archived redo logs to perform incomplete recovery. RMAN shuts down, starts up, and then opens the database with the RESETLOGS option to create the online logs.

Converting Filenames with CONFIGURE AUXNAME

This procedure assumes that you use the CONFIGURE AUXNAME command to rename the duplicate datafiles.

1. Follow the steps in "Preparing the Auxiliary Instance for Duplication: Basic Steps" on page 11-9, making sure to use an operating system utility to copy the parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure. Set all initialization parameters that end in _DEST or _PATH and specify a path name.

2. Add the following features when creating the RMAN commands to perform the duplication:
   - Prepare CONFIGURE AUXNAME commands for all datafiles, to be executed before database duplication.
   - If automatic auxiliary channels are not allocated, then allocate at least one auxiliary channel.
   - Use a LOGFILE clause to specify redo log groups and members for the duplicate database. (You do not have to use the same number of redo log groups or redo log group members in the duplicate database as you did in the target database.)
   - If you start the auxiliary instance with a client-side parameter file, then specify the PFILE parameter. The client-side parameter file must reside on the same host as the RMAN client used to perform the duplication.

The following example uses CONFIGURE AUXNAME to set new datafile names, uses automatic channels and a client-side initialization parameter file for the database duplication, and uses the LOGFILE clause to specify names and sizes for the online redo logs.

```
# configure the new desired filenames
 CONFIGURE AUXNAME FOR DATAFILE 1
        TO '/dup/oracle/oradata/trgt/system01.dbf';
 CONFIGURE AUXNAME FOR DATAFILE 2
        TO '/dup/oracle/oradata/trgt/undotbs01.dbf';
# ... add more CONFIGURE AUXNAME commands as needed

# run the DUPLICATE command
DUPLICATE TARGET DATABASE TO dupdb
# specify client-side parameter file for auxiliary instance if necessary
PFILE = /dup/oracle/dbs/initDUPDB.ora
.
.
.
# create at least two online redo log groups
 LOGFILE
    GROUP1
    (
      '/dup/oracle/oradata/trgt/redo01a.log',
      '/dup/oracle/oradata/trgt/redo01b.log',
      '/dup/oracle/oradata/trgt/redo01c.log';
    ) SIZE 200K,
    GROUP2
```

```
(
  '/dup/oracle/oradata/trgt/redo02a.log',
  '/dup/oracle/oradata/trgt/redo02b.log',
  '/dup/oracle/oradata/trgt/redo02c.log';
) SIZE 200K,
GROUP3
(
  '/dup/oracle/oradata/trgt/redo03a.log',
  '/dup/oracle/oradata/trgt/redo03b.log',
  '/dup/oracle/oradata/trgt/redo03c.log';
) SIZE 200K;
```

RMAN uses all incremental backups, archived redo log backups, and archived redo logs to perform incomplete recovery and then opens the database with the RESETLOGS option to create the online redo logs.

After the duplication is complete, clear the configured auxiliary names for the datafiles in the duplicate database, so that they are not overwritten by mistake. For example, enter the following:

```
# un-specify auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 CLEAR;
CONFIGURE AUXNAME FOR DATAFILE 2 CLEAR;
.
.
.
```

and so on.

Creating a Duplicate Database on the Local Host

When creating a duplicate database on the same host as the target database, follow the same procedure as for duplicating to a remote host with a different directory structure as described in "Duplicating a Database on a Remote Host with a Different Directory Structure" on page 11-14.

You can duplicate the database to the same Oracle home as the target database, but you must use a different database name from the target database, and convert the filenames by means of the same methods used for conversion on a separate host.

Caution: Do not use the NOFILENAMECHECK option when duplicating to the same Oracle home as the primary database. If you do, then the DUPLICATE command may overwrite the datafiles of the target database.

Duplicating a Database to an Automatic Storage Management Environment

The process for creating a duplicate database to an Automatic Storage Management or Oracle Managed Files location is similar to the procedure described in "Duplicating a Database on a Remote Host with a Different Directory Structure" on page 11-14. However, you must edit the initialization parameter file in the auxiliary instance to set the DB_CREATE_FILE_DEST parameter to refer to an ASM disk group, and eliminate parameters such as DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT which are used to control the naming of any files in the duplicate database which will be created in ASM during the duplication process.

For example, edit the file as follows to create the database files in the disk group named disk1:

```
*.DB_CREATE_FILE_DEST = '+disk1'
```

Database Duplication Examples

Here are some common variations on the process of duplicating a database.

Duplicating When the Datafiles Use Inconsistent Paths: Example

This example assumes the following:

- You are using recovery catalog database catdb.
- The target database trgt is on host1 and contains eight datafiles, which are spread out over multiple directories.
- You want to duplicate the target to database dupdb on remote host host2.
- host1 and host2 use different file systems.
- You want to store the datafiles in host2 in the /oradata1, /oradata2 ... through /oradata7 subdirectories.
- You want to exclude tablespace tools from the duplicate database, but keep all of the other tablespaces.
- You have used an operating system utility to copy the initialization parameter file from host1 to an appropriate location in host2.
- You have reset all initialization parameters that end in _DEST or _PATH and specify a path name.
- You want two online redo logs groups, each with two members of size 200 KB, which on the duplicate system will be stored in the directory "/duplogs".

- You have disk copies or backup sets stored on disk for all the datafiles and archived redo logs in the target database, and you have manually moved them to `host2` by means of an operating system utility.
- You have configured the default device to `sbt`. The media management device is accessible by `host2`.
- The auxiliary instance uses a server-side initialization parameter file in the default location (so the `PFILE` parameter is not necessary on the `DUPLICATE` command).

```
CONNECT TARGET /;
CONNECT CATALOG rman/cat@catdb;
CONNECT AUXILIARY SYS/oracle@dupdb;

# note that a RUN command is necessary because you can only execute SET NEWNAME
# within a RUN command
RUN
{
  # The DUPLICATE command uses an automatic sbt channel.
  # Because the target datafiles are spread across multiple directories,
  # run SET NEWNAME rather than DB_FILE_NAME_CONVERT
  SET NEWNAME FOR DATAFILE 1 TO '/oradata1/system01.dbf';
  SET NEWNAME FOR DATAFILE 2 TO '/oradata2/undotbs01.dbf';
  SET NEWNAME FOR DATAFILE 3 TO '/oradata3/cwmlite01.dbf';
  SET NEWNAME FOR DATAFILE 4 TO '/oradata4/drsys01';
  SET NEWNAME FOR DATAFILE 5 TO '/oradata5/example01.dbf';
  SET NEWNAME FOR DATAFILE 6 TO '/oradata6/indx01.dbf';
  # Do not set a newname for datafile 7, because it is in the tools tablespace,
  # and you are excluding tools from the duplicate database.
  SET NEWNAME FOR DATAFILE 8 TO '/oradata7/users01.dbf';
  DUPLICATE TARGET DATABASE TO dupdb
    SKIP TABLESPACE tools
    LOGFILE
      GROUP 1 ('/duplogs/redo01a.log',
               '/duplogs/redo01b.log') SIZE 200K REUSE,
      GROUP 2 ('/duplogs/redo02a.log',
               '/duplogs/redo02b.log') SIZE 200K REUSE;
}
```

Resynchronizing the Duplicate Database with the Target Database: Example

This example makes the same assumptions as in "Duplicating When the Datafiles Use Inconsistent Paths: Example" on page 11-20.

In this case, the assumption is that you want to update the duplicate database daily so that it stays current with the target database. Therefore the CONFIGURE command is used to set persistent new names for the datafiles, to use in the daily duplication process.

This script performs the onetime setup of the names for the data files, and should be run once.

```
# start RMAN and then connect to the databases
CONNECT TARGET /;
CONNECT CATALOG rman/cat@catdb;
CONNECT AUXILIARY SYS/oracle@dupdb;

# configure auxiliary names for the datafiles only once
CONFIGURE AUXNAME FOR DATAFILE 1 TO '/oradata1/system01.dbf';
CONFIGURE AUXNAME FOR DATAFILE 2 TO '/oradata2/undotbs01.dbf';
CONFIGURE AUXNAME FOR DATAFILE 3 TO '/oradata3/cwmlite01.dbf';
CONFIGURE AUXNAME FOR DATAFILE 4 TO '/oradata4/drsys01';
CONFIGURE AUXNAME FOR DATAFILE 5 TO '/oradata5/example01.dbf';
CONFIGURE AUXNAME FOR DATAFILE 6 TO '/oradata6/indx01.dbf';
# Do not set a newname for datafile 7, because it is in the tools tablespace,
# and you are excluding tools from the duplicate database.
CONFIGURE AUXNAME FOR DATAFILE 8 TO '/oradata7/users01.dbf';
```

This script is run daily to peform the duplication:

```
# start RMAN and then connect to the databases
CONNECT TARGET /;
CONNECT CATALOG rman/cat@catdb;
CONNECT AUXILIARY SYS/oracle@dupdb;

# Create the duplicate database. Issue the same command daily
# to re-create the database, thereby keeping the duplicate
# in sync with the target.
DUPLICATE TARGET DATABASE TO dupdb
SKIP TABLESPACE tools
  LOGFILE
    GROUP 1 ('/duplogs/redo01a.log',
             '/duplogs/redo01b.log') SIZE 200K REUSE,
    GROUP 2 ('/duplogs/redo02a.log',
             '/duplogs/redo02b.log') SIZE 200K REUSE;
```

Creating Duplicate of the Database at a Past Point in Time: Example

This duplication example assumes the following:

- The target database `trgt` and duplicate database `dupdb` are on different hosts but have exactly the same directory structure.
- You want to name the duplicate database files the same as the target files.
- You are not using a recovery catalog.
- You are using automatic channels for disk and sbt, which are already configured.
- You want to recover the duplicate database to one week ago in order to view the data in `prod1` as it appeared at that time.

```
CONNECT TARGET SYS/oracle@trgt
CONNECT AUXILIARY SYS/oracle@dupdb

DUPLICATE TARGET DATABASE TO dupdb
    NOFILENAMECHECK UNTIL TIME 'SYSDATE-7';
```

Duplicating with a Client-Side Parameter File: Example

If you use a client-side initialization parameter file to start the auxiliary instance, then it must reside on the same host as the RMAN client used to perform the duplication. Assume the following scenario:

- The target host is `host_tar` and the duplicate host is `host_dup`
- The client-side initialization parameter file on `host_dup` is named `/orahome/dbs/initTEST.ora`.
- The hosts `host_dup` and `host_tar` are linked by a network.

In this scenario, you can run the RMAN client (that is, run the `DUPLICATE` command in an RMAN session) either on `host_tar` or `host_dup`.

Running RMAN from host_dup

If you run the executable on `host_dup`, you can duplicate the database as follows:

```
DUPLICATE
  TARGET DATABASE TO dupdb
  DEVICE TYPE sbt
  PFILE='/orahome/dbs/initTEST.ora';
```

Because the initialization parameter file used by the auxiliary instance resides on the same node as the RMAN client, you can reference the local filename of the parameter file.

Running RMAN from host_tar

In this scenario, you run RMAN on the same host as the target database rather than on the host with the duplicate database. Hence, the client-side initialization parameter file needed by the DUPLICATE command is not located on the same node as the RMAN client. You must transfer the parameter file from host_dup to host_tar, or remotely mount the directory containing the parameter file by some means such as NFS, so that it can be accessed from the target host.

Copying the Parameter File from host_dup to host_tar In this scenario, you manually copy the file from one host to another. In Unix systems you could use the cp command:

```
% cp /net/host_dup/orahome/dbs/initTEST.ora /net/host_tar/tmp
```

Then, you can start RMAN on host_tar and perform the duplication with the following command:

```
% rman TARGET SYS/oracle@trgt AUXILIARY SYS/oracle@dupdb
RMAN> DUPLICATE TARGET DATABASE TO dupdb
        DEVICE TYPE sbt PFILE='/net/host_tar/tmp/initTEST.ora';
RMAN> EXIT
```

Mounting the host_dup File System on host_tar In this scenario, you mount the host_dup file system on the host_tar file system by using /net/host_dup as the mount point. The /net/host_dup/initTEST.ora filename on host_tar points to the /orahome/dbs/initTEST.ora file residing on host_dup. Then, you can run the duplication as follows:

```
DUPLICATE
  TARGET DATABASE TO dupdb
  DEVICE TYPE sbt
  PFILE='/net/host_dup/initTEST.ora';
```

12

Migrating Databases To and From ASM with Recovery Manager

This chapter describes how to migrate a database into and out of an ASM disk group using Recovery Manager. It covers the following topics:

- Migrating a Database into ASM
- Migrating the Flash Recovery Area to ASM
- Migrating a Database from ASM to Non-ASM Storage
- PL/SQL Scripts Used in Migrating to ASM Storage

Migrating a Database into ASM

To take advantage of Automatic Storage Management with an existing database you must migrate that database into ASM. This migration is performed using Recovery Manager (RMAN) even if you are not using RMAN for your primary backup and recovery strategy.

A database can be moved from non-ASM disk storage directly into ASM, or you can back the database up to tape and then from tape backups move it into ASM. Moving the database to tape backup and then into ASM is recommended if your database is so large that you cannot have copies of the database on non-ASM disk storage and ASM disk storage simultaneously. You can back the database up to tape, convert your non-ASM disk storage into an ASM disk group, and then restore from tape to the ASM disk group.

Limitation on ASM Migration with Transportable Tablespaces

The procedure described here does not work for transportable (foreign) tablespaces. Such tablespaces needs to be made read-write and imported into the database, before they can be migrated into ASM using this procedure.

Preparing to Migrate a Database to ASM

There are several steps required to prepare your database for migration and collect useful information you will need later, before you start the actual migration process.

Determine Your DBID

If you are not using a recovery catalog, you may need to know your DBID. You must restore your control file from autobackup during the migration process, and you will need to set your DBID before you restore the control file.

Your DBID should be part of the permanent records you keep about your database. If you do not have it in your records, the easiest way to find out your DBID is to connect the RMAN client to the database to be migrated. RMAN displays the DBID whenever it starts up. For example:

```
% rman TARGET /
Recovery Manager: Release 10.1.0.2.0 - Production

Copyright (c) 1995, 2003, Oracle.  All rights reserved.

connected to target database: RDBMS (DBID=774627068)
```

```
RMAN> exit
```

Make a note of this value.

Determine Names of Database Files

Obtain the filenames of the control files, datafiles, and online redo logs for your database. This information will useful if you decide to migrate back to old (non-ASM) storage later. Information about datafiles is available by querying `V$DATAFILE`, and the control file names can be found in the `CONTROL_FILES` initialization parameter.

Generate RMAN Command File to Undo ASM Migration

If you need to migrate your database back to non-ASM storage later, this process will be simplified if you generate an RMAN command file now with the necessary commands to perform this migration. Even if you make changes to your database later, such as adding datafiles, the command file you create now will serve as a useful starting point.

There is a PL/SQL script described in "Generating ASM-to-Non-ASM Storage Migration Script" on page 12-14 which will generate the necessary RMAN commands for you. Run this script and save the output as part of the permanent records you keep for your database.

Disk-Based Migration of a Database to ASM

If you have enough disk space that you can have both your entire non-ASM database and your ASM disk group on disk at the same time, you can do the migration directly without using tapes.

Once you have completed the preparations in "Preparing to Migrate a Database to ASM" on page 12-2, begin the migration procedure.

The procedure differs slightly between primary and standby databases. A number of the steps described in this procedure apply only in one or the other case. There are also a few steps where the procedure is different depending upon whether you are using a recovery catalog. The steps that vary are identified as necessary in the description of the process.

To perform the migration, carry out the following steps:

1. Disable change tracking.

   ```
   SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
   ```

2. If this is standby database, stop managed recovery mode.

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

3. Shut down the database consistently. For example:

```
SQL> SHUTDOWN IMMEDIATE
```

4. Modify the initialization parameter file of the target database as follows:

- Set DB_CREATE_FILE_DEST and DB_CREATE_ONLINE_LOG_DEST_*n* (where *n*=1-4) to refer to the desired ASM disk groups.
- If the database uses a server parameter file (SPFILE), then remove the CONTROL_FILES parameter that specifies locations of the control file. The control file will be moved to the DB_CREATE_* destination and the server parameter file will be automatically updated. If you are using a client-side parameter file (PFILE), then set the CONTROL_FILES parameter to the ASM alias for the control file name (for example, CONTROL_FILES=+disk_group/cf1).

5. Startup the database in nomount mode:

```
RMAN> STARTUP NOMOUNT;
```

6. Restore the control file into new locations from location specified in the old SPFILE or PFILE:

```
RMAN> RESTORE CONTROLFILE FROM 'filename_of_old_control_file';
```

7. Mount the database. For example:

```
RMAN> ALTER DATABASE MOUNT;
```

8. Copy the database into the ASM disk group using the following command:

```
RMAN> BACKUP AS COPY DATABASE FORMAT '+disk_group';
```

   You may be able to speed up the copy process by increasing RMAN parallelism. For example:

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 4;
RMAN> BACKUP AS COPY DATABASE FORMAT '+disk_group';
```

9. Switch all datafiles into new ASM disk group

```
RMAN> SWITCH DATABASE TO COPY;
```

All datafiles are now in ASM.

If this database a standby database, then **do not** open the database or start managed recovery at this time.

If this is a primary database (that is, not a standby database), then you can open the database.

```
RMAN> ALTER DATABASE OPEN;
```

Note that there is no need to perform an `OPEN RESETLOGS`.

10. If you are migrating a primary database, then move your online logs into ASM at this time. For each online redo log, create a new one in ASM, archive the current redo log, and then delete the old non-ASM log. For a PL/SQL script that can perform this task for you, see "Migrating Online Redo Logs to ASM Storage" on page 12-15.

    If this is a standby database, then online logs do not exist (only standby online logs exist) and therefore cannot be renamed at this point. However, you must delete any online log files that might have been created if the database was used as a primary database in the past. Delete these files using the operating system delete command.

    When the standby database is activated, online logs will automatically be added as ASM files. For each standby online redo log file, create a new one in ASM, and delete an old one from the non-ASM storage. For a PL/SQL script that can perform this task for you, see "Migrating Standby Online Redo Log Files to ASM Storage" on page 12-14.

At this point the migration is complete. The original datafiles, still in non-ASM storage, are cataloged as datafile copies in the RMAN repository. You can use them as backups, or reclaim the disk space by deleting them.

If you were using change tracking for incremental backups, you can re-enable it now. For example:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

If you decide to migrate back to old storage, you can switch back to the original datafiles, using the script created in "Preparing to Migrate a Database to ASM" on page 12-2. If you have not yet deleted your original datafiles, you can also use the `SWITCH DATABASE TO COPY` command to switch back rather than going through the whole migration process.

Cleanup of Non-ASM Files After ASM Migration

You can delete the old database files to free disk space. The RMAN repository has records of the old datafiles, so you can delete these with an RMAN command. The old control files and online redo logs, however, are not in the repository and must be deleted with host operating system commands. This example shows how to perform this under Unix, using the `rm` command for the online redo logs and control files:

```
# delete datafiles
RMAN> DELETE COPY OF DATABASE;
RMAN> HOST 'rm old_online_redo_logs';
RMAN> HOST 'rm old_control_files';
```

Using Tape Backups to Migrate a Database to ASM

This alternative procedure is useful when you do not have enough disk space to hold both the non-ASM version of your database and the ASM disk group that will hold your database after the migration. The database is backed up from non-ASM storage to tape using RMAN, then restored from tape into ASM storage.

Once you have completed the preparations in "Preparing to Migrate a Database to ASM" on page 12-2, begin the migration procedure.

Performing Migration of a Database to ASM Storage using RMAN Tape Backup

To migrate your non-ASM database into ASM storage using a tape backup, use the following procedure:

1. Backup the whole database to tape using RMAN. For example:

   ```
   RMAN> BACKUP DEVICE TYPE SBT DATABASE;
   ```

   After backup is done, disable change tracking. For example:

   ```
   SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
   ```

   If this is standby database, stop managed recovery mode.

   ```
   SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
   ```

Shut down the database consistently. For example:

```
SQL> SHUTDOWN IMMEDIATE
```

At this point you can delete the old database files and create your ASM disk groups.

2. Modify the initialization parameter file of the target database as follows:

- Set DB_CREATE_FILE_DEST and DB_CREATE_ONLINE_LOG_DEST_*n* (where *n*=1-4) to refer to the desired ASM disk groups.
- If the database uses a server parameter file (SPFILE), then remove the CONTROL_FILES parameter that specifies locations of the control file. The control file will be moved to the DB_CREATE_* destination and the server parameter file will be automatically updated.

  If you are using a client-side parameter file (PFILE), then set the CONTROL_FILES parameter to the ASM alias for the control file name (for example, CONTROL_FILES=+disk_group/cf1).

3. Startup the database in nomount mode. For example:

```
RMAN> STARTUP NOMOUNT;
```

4. Restore the control file from backup into its new locations. If you are using a reovery catalog, then issue this command:

```
RMAN> RESTORE CONTROLFILE;
```

   If you are not using a recovery catalog, then you will need to set your DBID and allocate channels to perform the restore of your control file. For example:

```
RMAN> RESTORE CONTROLFILE FROM AUTOBACKUP;
```

5. Mount the database. For example:

```
RMAN> ALTER DATABASE MOUNT;
```

6. Now restore database into new location, using RMAN commands. You can enter these manually, as shown in this example:

```
RMAN> RUN {
2>  SET  NEWNAME FOR DATAFILE 1 TO NEW;
3>  SET  NEWNAME FOR DATAFILE 2 TO NEW;
4> ...add one line for each datafile
5>  RESTORE DATABASE;
6>  SWITCH ALL;
6>  }
```

   Be sure to include a SET NEWNAME FOR DATAFILE *filenumber* TO NEW statement for each datafile in the database.

The RESTORE DATABASE command instructs RMAN to restore into location DB_CREATE_FILE_DEST, which has now been updated to refer to the ASM disk group. New names are generated for each datafile automatically.

If the database has a large number of datafiles, it is more convenient to use the following PL/SQL script to generate the required RMAN commands for the RESTORE operation:

```
set serveroutput on;
 declare
    cursor df is select file# from v$datafile;
 begin
    dbms_output.put_line('run');
    dbms_output.put_line('{');
    for dfrec in df loop
    dbms_output.put_line('set newname for datafile ' ||
                         dfrec.file# || ' to new;');
 end loop;
 dbms_output.put_line('restore database;');
 dbms_output.put_line('switch all;');
 dbms_output.put_line('}');
 end;
```

Running this PL/SQL code causes the SQL*Plus client to display an RMAN script which you can save to a file and run as a command file in the RMAN client.

After this operation all datafiles are in ASM.

Now you must create new tempfiles for the temporary tablespaces. For each temporary tablespace, execute the following command:

```
RMAN> SQL "ALTER TABLESPACE tablespacename ADD TEMPFILE;"
```

If this a primary database, then recover the database and perform an OPEN RESETLOGS on the database.

```
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Note that you must use the RESETLOGS option because the control file is restored from backup.

If you were using change tracking for incremental backups, you can re-enable it now. For example:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

If you are migrating a standby database, do not open the database at this time. If your standby database has standby online logs stored in the flash recovery area, then you must move standby online log files into ASM storage. For each standby online redo log file, you must create a new one in ASM, and delete an old one from the non-ASM storage. For a PL/SQL script that can perform this task for you, see "PL/SQL Scripts Used in Migrating to ASM Storage" on page 12-14.

Migrating the Flash Recovery Area to ASM

The following procedure assumes that you have a flash recovery area in non-ASM disk storage and you need to move it to an ASM disk group, possibly preserving its contents.

1. If logging for Flashback Database is enabled, then disable it. This is needed because the logs for Flashback Database are located in the flash recovery area. For example:

   ```
   SQL> ALTER DATABASE FLASHBACK OFF;
   ```

2. If this database is a primary database and your online logs, control file or archived redo logs are in the flash recovery area, then perform a consistent shutdown of your database. For example:

   ```
   SQL> SHUTDOWN IMMEDIATE
   ```

   If this database is a standby database and your standby online logs, controlfile, or archive logs are in recovery area, then stop managed recovery mode and shutdown database.

3. Modify the initialization parameter file of the target database as follows:
   - Set `DB_RECOVERY_FILE_DEST` to the desired ASM disk group.
   - Modify `DB_RECOVERY_FILE_DEST_SIZE` if you need to change the size of the flash recovery area.

4. If you shut down the database in step 2, then bring the database to a NOMOUNT state. For example:

   ```
   RMAN> STARTUP NOMOUNT
   ```

   If the old recovery area has copy of the current controlfile, then restore controlfile from the old `DB_RECOVERY_FILE_DEST` and mount the database again.

   ```
   RMAN> RESTORE CONTROLFILE FROM 'filename_of_old_control_file';
   ```

```
RMAN> ALTER DATABASE MOUNT;
```

5. If you are using tape backups, then you should back up the entire flash recovery area to tape at this time. For example:

   ```
   RMAN> BACKUP RECOVERY AREA;
   ```

   You can also use the `DELETE INPUT` option when backing up the flash recovery area to tape, if you want to immediately free the non-ASM space previously used to store flash recovery area files:

   ```
   RMAN> BACKUP RECOVERY AREA DELETE INPUT;
   ```

   Note: If you do not have tape, you should not delete the flash recovery area storage.

6. If you were using flashback logging before to support flashback database, you can re-enable it now. For example:

   ```
   SQL> ALTER DATABASE FLASHBACK ON;
   ```

   Now, optionally, you can move your backups from old recovery area to the new location. To move the existing backupsets and archived redo log files, use these two commands:

   ```
   RMAN> BACKUP AS COPY ARCHIVELOG ALL DELETE INPUT;
   RMAN> BACKUP DEVICE TYPE DISK BACKUPSET ALL DELETE INPUT;
   ```

   Then you must move your datafile copies. For each datafile copy in the old recovery area, use this command:

   ```
   RMAN> BACKUP AS COPY DATAFILECOPY "name" DELETE INPUT;
   ```

   where *name* is the path to the datafile copy in the old recovery area location.

   If the old recovery area location contains a large number of files, you can use the following PL/SQL script to generate the RMAN commands required to relocate the files:

   ```
   set serveroutput on;
   declare
     cursor dfc is select name from v$datafile_copy
                   where status = 'A'
                     and is_recovery_dest_file = 'YES';
   begin
     dbms_output.put_line('run');
     dbms_output.put_line('{');
   ```

```
    dbms_output.put_line('backup as copy archivelog all delete input;');
    dbms_output.put_line('backup device type disk backupset all delete
input;');
  for dfcrec in dfc loop
    dbms_output.put_line('backup as copy datafilecopy ''' ||
                         dfcrec.name || '''delete input;');
  end loop;
  dbms_output.put_line('}');
end;
```

7. If this database is a standby database, then do not open the database at this point. If this database is a physical standby database, then restart managed recovery mode.

   If this is a primary database (that is, not a standby database), then open the database:

   ```
   RMAN> alter database open;
   ```

8. If this is a standby database, then the online logs cannot be renamed at this point. You must delete the files containing the online redo logs at the operating system level. When the standby database is activated, new online logs will automatically be added as ASM files.

   If this is a primary database and you had online redo log files in the flash recovery area, then you should set up your database to store the online redo logs in the ASM disk group. For each online redo log group, you must create a new online redo log in the ASM disk group, archive the current logs, and delete the old log member. For a PL/SQL script that can perform this task for you, see "Migrating Online Redo Logs to ASM Storage" on page 12-15.

9. If this is a standby database and you had standby online logs in the recovery area, you should move standby online logs into ASM. For a PL/SQL script that can perform this task for you, see "Migrating Standby Online Redo Log Files to ASM Storage" on page 12-16.

At this point the migration of the flash recovery area is complete.

Migrating a Database from ASM to Non-ASM Storage

Migrating a database back from ASM storage to non-ASM storage is similar to the original migration to ASM. The process described here assumes that you can perform the migration through tape. It is very similar to the process described in "Using Tape Backups to Migrate a Database to ASM" on page 12-6. You can also

migrate from ASM to non-ASM storage using only disk, using a process similar to the one used to migrate into ASM storage using only disk.

1. If you are not using a recovery catalog, then determine your DBID, as described in "Determine Your DBID" on page 12-2. Write it down, because you will use it in restoring your control file into non-ASM storage.
2. Backup the database to tape. For example:

   ```
   RMAN> BACKUP DEVICE TYPE SBT DATABASE;
   ```

3. After backup is done, disable change tracking. For example:

   ```
   SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
   ```

4. If this is a standby database, stop managed recovery mode.
5. Perform a consistent shutdown of the database. For example:

   ```
   SQL> SHUTDOWN IMMEDIATE
   ```

6. Delete the ASM disk groups.
7. Modify the initialization parameter file of the database as follows:
   - Remove `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_`*n* parameters.
   - Add the `CONTROL_FILES` parameter that specifies locations of the control file.
8. Startup the database in NOMOUNT mode. For example:

   ```
   RMAN> STARTUP NOMOUNT;
   ```

9. Restore the control file into new locations from backup.

   If you are not using a recovery catalog, then you must use a control file autobackup. Set your DBID and restore the control file, as follows:

   ```
   RMAN> SET DBID 320066378;
   RMAN>  RUN {
      ALLOCATE CHANNEL ctape DEVICE TYPE SBT
         PARMS='...';
      ALLOCATE CHANNEL cdisk DEVICE TYPE DISK;
      RESTORE CONTROLFILE FROM AUTOBACKUP;
   }
   ```

If you use a recovery catalog, then you do not need to use a control file autobackup, because the recovery catalog contains a record of the control file backup location. Restore the control file, as follows:

```
RMAN>  RUN {
   ALLOCATE CHANNEL ctape DEVICE TYPE SBT
      PARMS='...';
   ALLOCATE CHANNEL cdisk DEVICE TYPE DISK;
   RESTORE CONTROLFILE;
}
```

10. Mount the database. For example:

```
RMAN> ALTER DATABASE MOUNT;
```

11. Now restore database into new location. Use the RMAN script generated by the procedure described in "Generating ASM-to-Non-ASM Storage Migration Script" on page 12-14. As noted in that section, you may need to alter the script if you have made structural changes to your database since you first migrated to ASM, or if you want to the location where you wish to store the datafiles. After this operation all datafiles will be in the locations specified in the file.

12. Now you must create new tempfiles for the temporary tablespaces. For each temporary tablespace, execute the following command:

```
RMAN> SQL "ALTER TABLESPACE tablespacename ADD TEMPFILE tempfilename;"
```

13. If this a standby database, then **do not** open the database at this time.

    If this is a primary database, then recover and open the database. Note that you must perform an OPEN RESETLOGS because the control file is restored from backup.

```
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

14. If this is a primary database, then move your online logs back into old location. For each online redo log group, carry out the following steps in SQL*Plus:

```
ALTER DATABASE ADD LOGFILE SIZE BYTES FILENAME new_name;
ALTER DATABASE ARCHIVE LOG CURRENT;
ALTER DATABASE DROP LOGFILE old_name;
```

    If this is a standby database, then you should move standby online logs back to the old location. For each standby online log, execute the following commands in SQL*Plus:

```
ALTER DATABASE ADD STANDBY LOGFILE SIZE BYTES FILENAME new_name;
ALTER DATABASE DROP STANDBY LOGFILE old_name;
```

15. If this is a standby database, then start managed recovery mode at this time.

At this point, the migration from ASM to non-ASM storage is complete. You may want to enable change tracking for incremental backups, if you were using it before the migration. For example:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

PL/SQL Scripts Used in Migrating to ASM Storage

The following PL/SQL scripts perform tasks which arise in more than one of the migration scenarios described in this chapter.

Generating ASM-to-Non-ASM Storage Migration Script

You can use the following PL/SQL script to generate a series of RMAN commands that you can use to migrate your database back from ASM to non-ASM disk storage.

```
set serveroutput on;
declare
    cursor df is select file#, name from v$datafile;
begin
    dbms_output.put_line('run');
    dbms_output.put_line('{');
    for dfrec in df loop
        dbms_output.put_line('set newname for datafile ' ||
            dfrec.file# || ' to ''' || dfrec.name ||''' ;');
    end loop;
    dbms_output.put_line('restore database;');
    dbms_output.put_line('switch all;');
    dbms_output.put_line('}');
end;
```

Running this PL/SQL code causes the SQL*Plus client to display an RMAN script which you can save to a file and later run as a command file in the RMAN client to migrate your datafiles back out of ASM storage.

The script queries V$DATAFILE to find out the filenames for your datafiles before they are moved to ASM. Running this script later will restore the same set of datafiles to their pre-ASM locations. If you decide to store the files in a different

disk location when moving them out of ASM, update the generated RMAN commands to use different destination filenames. If you add datafiles to your database, edit the script to include SET NEWNAME commands to specify locations for the new datafiles. If you delete datafiles, remove the corresponding commands from the migration script.

Migrating Online Redo Logs to ASM Storage

The following PL/SQL script can be used to migrate your online redo log groups into ASM, as part of migrating a database or a flash recovery area into ASM. For each online redo log group, the script adds a log file stored in ASM, archives the current redo logs, and then drops the non-ASM log file.

Save this script into a file and run it from within SQL*Plus to migrate the online logs.

```
declare
   cursor orlc is select lf.member, l.bytes
                    from v$log l, v$logfile lf
                   where l.group# = lf.group#
                     and lf.type = 'ONLINE'
                   order by l.thread#, l.sequence#;
   type numTab_t is table of number index by binary_integer;
   type charTab_t is table of varchar2(1024) index by binary_integer;
   byteslist numTab_t;
   namelist  charTab_t;
   procedure migrateorlfile(name IN varchar2, bytes IN number) is
      retry    number;
      stmt     varchar2(1024);
      als      varchar2(1024) := 'alter system switch logfile';
   begin
      select count(*) into retry from v$logfile;
      stmt := 'alter database add logfile size ' || bytes;
      execute immediate stmt;
      stmt := 'alter database drop logfile ''' || name || '''';
      for i in 1..retry loop
         begin
            execute immediate stmt;
            exit;
         exception
            when others then
            if i > retry then
               raise;
            end if;
```

```
            execute immediate als;
         end;
       end loop;
    end;
begin
    open orlc;
    fetch orlc bulk collect into namelist, byteslist;
    close orlc;
    for i in 1..namelist.count loop
       migrateorlfile(namelist(i), byteslist(i));
    end loop;
end;
```

Migrating Standby Online Redo Log Files to ASM Storage

The following PL/SQL script can be used to migrate your standby online redo log files into ASM, as part of migrating your whole database into ASM.

Save this script into a file and run it from within SQL*Plus to migrate the standby online logs.

```
declare
   cursor srlc is select lf.member, l.bytes
                    from v$standby_log l, v$logfile lf
                   where l.group# = lf.group#
                     and lf.type = 'STANDBY';
   type numTab_t is table of number index by binary_integer;
   type charTab_t is table of varchar2(1024) index by binary_integer;
   byteslist numTab_t;
   namelist  charTab_t;
   procedure migratesrl(name IN varchar2, bytes IN number) is
      stmt     varchar2(1024);
   begin
      stmt := 'alter database add standby logfile size ' || bytes;
      execute immediate stmt;
      stmt := 'alter database drop standby logfile ''' || name || '''';
      execute immediate stmt;
   end;
begin
   open srlc;
   fetch srlc bulk collect into namelist, byteslist;
   close srlc;
   for i in 1..namelist.count loop
      migratesrl(namelist(i), byteslist(i));
   end loop;
```

```
end;
```

13

Managing the Recovery Catalog

This chapter describes how to manage an RMAN recovery catalog, which holds RMAN repository data for one or more databases in a separate database schema, in addition to using the control files of the databases.

This chapter contains these topics:

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how to manage the RMAN control file repository without a recovery catalog

Creating a Recovery Catalog

Creating a recovery catalog is a three-step process: you must configure the database that will contain the recovery catalog, create the recovery catalog owner, and then create the recovery catalog itself.

Configuring the Recovery Catalog Database

When you use a recovery catalog, RMAN requires that you maintain a recovery catalog schema. The recovery catalog is stored in the default tablespace of the schema. Note that SYS cannot be the owner of the recovery catalog.

Decide which database you will use to install the recovery catalog schema, and also how you will back up this database. You should not install the catalog in the target database: this tactic defeats the purpose of the catalog. Also, decide whether to operate the catalog database in ARCHIVELOG mode, which is recommended.

Planning the Size of the Recovery Catalog Schema

You must allocate space to be used by the catalog schema. The size of the recovery catalog schema depends upon the number of databases monitored by the catalog, and the number and size of RMAN scripts stored in the catalog. The schema also grows as the number of archived redo log files and backups for each database grows.

For an example, assume that the `trgt` database has 100 files, and you back up the database once a day, producing 50 backup sets containing 1 backup piece each. If you assume that each row in the backup piece table uses the maximum amount of space, then one daily backup will consume less than 170 KB in the recovery catalog. So, if you back up once a day for a year, then the total storage in this period is about 62 MB. Assume approximately the same amount for archived logs. Hence, the worst case is about 120 MB for a year for metadata storage.

For a more typical case in which only a portion of the backup piece row space is used, 15 MB for each year is a realistic estimate.

If you plan to register multiple databases in your recovery catalog, remember to add up the space required for each one based on the previous calculation to arrive at a total size for the default tablespace of the recovery catalog schema.

Allocating Disk Space for the Recovery Catalog Database

If you are creating your recovery catalog in an already-existing database, add enough room to hold the default tablespace to the recovery catalog schema. If you

are creating a new database to hold your recovery catalog, then, in addition to the space for the recovery catalog schema itself, you must allow space for other files in the recovery catalog database:

- SYSTEM tablespace
- Temporary tablespaces
- Rollback segment tablespaces
- Online redo log files

Most of the space used in the recovery catalog database is devoted to supporting tablespaces, for example, the SYSTEM, temporary, and rollback or undo tablespaces. Table 13–1 describes typical space requirements.

Table 13–1 Typical Recovery Catalog Space Requirements for 1 Year

Type of Space	Space Requirement
SYSTEM tablespace	90 MB
Temp tablespace	5 MB
Rollback or undo tablespace	5 MB
Recovery catalog tablespace	15 MB for each database registered in the recovery catalog
Online redo logs	1 MB each (3 groups, each with 2 members)

Caution: Ensure that the recovery catalog and target databases do *not* reside on the same disk. If a disk containing both your recovery catalog and your target databases failed, your recovery process would be much more complicated. If possible, take other measures as well to eliminate common points of failure between your recovery catalog database and the databases you are backing up.

Creating the Recovery Catalog Owner

After choosing the recovery catalog database and creating necessary space, you are ready to create the owner of the recovery catalog and grant this user necessary privileges.

Assume the following background information for the instructions in the following sections:

- User SYS with password `oracle` has SYSDBA privileges on the recovery catalog database `catdb`.
- A tablespace called `tools` in the recovery catalog database `catdb` stores the recovery catalog. Note that to use an RMAN reserved word as a tablespace name, you must enclose it in quotes and put it in uppercase. (Refer to *Oracle Database Recovery Manager Reference* for a list of RMAN reserved words.)
- A tablespace called `temp` exists in the recovery catalog database.
- The database is configured in the same way as all normal databases, for example, `catalog.sql` and `catproc.sql` have successfully run.

To create the recovery catalog schema in the recovery catalog database:

1. Start SQL*Plus and then connect with administrator privileges to the database containing the recovery catalog. For example, enter:

   ```
   CONNECT SYS/oracle@catdb AS SYSDBA
   ```

2. Create a user and schema for the recovery catalog. For example, enter:

   ```
   CREATE USER rman IDENTIFIED BY cat
     TEMPORARY TABLESPACE temp
     DEFAULT TABLESPACE tools
     QUOTA UNLIMITED ON tools;
   ```

3. Grant the RECOVERY_CATALOG_OWNER role to the schema owner. This role provides the user with all privileges required to maintain and query the recovery catalog.

   ```
   SQL> GRANT RECOVERY_CATALOG_OWNER TO rman;
   ```

Creating the Recovery Catalog

After creating the catalog owner, create the catalog tables with the RMAN CREATE CATALOG command. The command creates the catalog in the default tablespace of the catalog owner.

To create the recovery catalog:

1. Connect to the database that will contain the catalog as the catalog owner. For example, enter the following from the operating system command line:

   ```
   % rman CATALOG rman/cat@catdb
   ```

You can also connect from the RMAN prompt:

```
% rman
RMAN> CONNECT CATALOG rman/cat@catdb
```

2. Run the CREATE CATALOG command to create the catalog. The creation of the catalog can take several minutes. If the catalog tablespace is this user's default tablespace, then you can run this command:

   ```
   CREATE CATALOG;
   ```

   If you specify the tablespace name in the CREATE CATALOG command, *and* if the tablespace name is an RMAN reserved word, then it *must* be uppercase and enclosed in quotes. For example:

   ```
   CREATE CATALOG TABLESPACE 'CATALOG';
   ```

3. Optionally, start SQL*Plus and query the recovery catalog to see which tables were created:

   ```
   SQL> SELECT TABLE_NAME FROM USER_TABLES;
   ```

   See Also: *Oracle Database SQL Reference* for the SQL syntax for the GRANT and CREATE USER statements, and *Oracle Database Recovery Manager Reference* for CREATE CATALOG command syntax

Managing Target Database Records in the Recovery Catalog

This section describes how to register, unregister, and reset target database records in the recovery catalog.

Registering a Database in the Recovery Catalog

The first step in using a recovery catalog with a target database is registering the database in the recovery catalog. Use the following procedure:

1. After making sure the recovery catalog database is open, connect RMAN to the target database and recovery catalog database. For example, issue the following to connect to the catalog database `catdb` as user `rman` (who owns the catalog schema):

   ```
   % rman TARGET / CATALOG rman/cat@catdb
   ```

2. If the target database is not mounted, then mount or open it:

   ```
   RMAN> STARTUP MOUNT;
   ```

3. Register the target database in the connected recovery catalog:

```
RMAN> REGISTER DATABASE;
```

RMAN creates rows in the catalog tables to contain information about the target database, then copies all pertinent data about the target database from the control file into the catalog, synchronizing the catalog with the control file.

Verify that the registration was successful by running REPORT SCHEMA:

```
RMAN> REPORT SCHEMA;

Report of database schema
File K-bytes    Tablespace       RB segs Datafile Name
---- ---------- ---------------- ------- ------------------
1        307200 SYSTEM             NO    /oracle/oradata/trgt/system01.dbf
2         20480 UNDOTBS            YES   /oracle/oradata/trgt/undotbs01.dbf
3         10240 CWMLITE            NO    /oracle/oradata/trgt/cwmlite01.dbf
4         10240 DRSYS              NO    /oracle/oradata/trgt/drsys01.dbf
5         10240 EXAMPLE            NO    /oracle/oradata/trgt/example01.dbf
6         10240 INDX               NO    /oracle/oradata/trgt/indx01.dbf
7         10240 TOOLS              NO    /oracle/oradata/trgt/tools01.dbf
8         10240 USERS              NO    /oracle/oradata/trgt/users01.dbf
```

Cataloging Older Files in the Recovery Catalog

If you have datafile copies, backup pieces or archive logs on disk, you can catalog them in the recovery catalog using the CATALOG command. When using a recovery catalog, cataloging older backups that have aged out of the control file lets RMAN use the older backups during restore operations. For example:

```
RMAN> CATALOG DATAFILECOPY '/disk1/old_datafiles/01_01_2003/users01.dbf';
RMAN> CATALOG ARCHIVELOG '/disk1/arch_logs/archive1_731.dbf',
     '/disk1/arch_logs/archive1_732.dbf';
RMAN> CATALOG BACKUPPIECE '/disk1/backups/backup_820.bkp';
```

You can also catalog multiple backup files in a directory at once, using the CATALOG START WITH command, as shown in this example:

```
RMAN> CATALOG START WITH '/disk1/backups/';
```

RMAN lists the files to be added to the RMAN repository and prompts for confirmation before adding the backups.

Be careful when creating your prefix for CATALOG START WITH. RMAN scans all paths for all files on disk which begin with your specified prefix. The wrong prefix may include more files than you intend. For example, a group of directories /disk1/backups , /disk1/backups-year2003, /disk1/backupsets, and /disk1/backupsets/test and so on, all contain backup files. The command

```
RMAN> CATALOG START WITH '/disk1/backups';
```

catalogs all files in all of these directories, because /disk1/backups is a prefix for the paths for all of these directories. In order to catalog only backups in the /disk1/backups directory, the correct command would be:

```
RMAN> CATALOG START WITH '/disk1/backups/';
```

To determine whether log records have aged out of the control file, compare the number of logs on disk with the number of records in V$ARCHIVED_LOG.

Cataloging Oracle7 Datafile Copies in the Recovery Catalog In general, only Oracle8 and higher files can be cataloged. Datafile copies from Oracle7 databases can also be cataloged, if they do not require the application of Oracle7 redo before they can be opened. Datafile copies made in the following circumstances satisfy this requirement:

- Datafile copies made when the database was shut down consistently. The database must not have been opened again before migration to a higher version of Oracle.
- Datafile copies made after a tablespace became offline normal or read-only. The tablespaces must not have been brought online or made read / write again before migration to a higher version of Oracle.

See Also:

- *Oracle Database Recovery Manager Reference* for REGISTER syntax
- *Oracle Database Upgrade Guide* for issues relating to database migration

Registering Multiple Databases in a Recovery Catalog

You can register multiple target databases in a single recovery catalog, if they do not have duplicate DBIDs. RMAN uses the DBID to distinguish one database from another.

In general, if you use the DUPLICATE RMAN command or CREATE DATABASE SQL statement, the database created is assigned a unique DBID. If you create a database by some other means, such as a user-managed copy, then the new database may have the same DBID as the one from which it was copied. You will not be able to register both databases in the same recovery catalog until you change the DBID of the copied database using the DBNEWID utility.

Note that you can register a single target databases in multiple recovery catalogs.

See Also:

- *Oracle Database Recovery Manager Reference* for DUPLICATE syntax
- *Oracle Database Utilities* to learn how to use the DBNEWID utility to change the DBID
- *Oracle Database Upgrade Guide* for issues relating to database migration

Unregistering a Target Database from the Recovery Catalog

RMAN can unregister a database as well as register it. The UNREGISTER DATABASE command is used to unregister a database from the recover catalog.

Make sure this procedure is what you intend, because if you make a mistake, then you must re-register the database. You will lose any RMAN repository records older than the CONTROLFILE_RECORD_KEEP_TIME setting in the target database control file.

To unregister a database:

1. Start RMAN and connect to the target database. For example, enter:

```
% rman TARGET / CATALOG rman/cat@catdb

connected to target database: RDBMS (DBID=1237603294)
connected to recovery catalog database
```

   Make a note of the DBID as displayed by RMAN. If there is more than one database registered in the recovery catalog, you will need the DBID to uniquely identify the database.

   It is not necessary to connect to the target database, but if you do not, then you must specify the name of the target database in the UNREGISTER command. If more than one database has the same name in the recovery catalog, then you

must create a RUN block around the command and use SET DBID to set the DBID for the database .

2. It may be useful to list all of the backups recorded in the recovery catalog using `LIST BACKUP SUMMARY` and `LIST COPY SUMMARY`.
3. If your intention is to actually delete all backups of the database completely, rather than just removing the database from the recovery catalog and relying on the control file to store the RMAN repository for this database, then run `DELETE` statements to delete all existing physical backupsets and image copies. For example:

   ```
   DELETE BACKUP DEVICE TYPE sbt;
   DELETE BACKUP DEVICE TYPE DISK;
   DELETE COPY;
   ```

   RMAN will list the backups that it intends to delete and prompt for confirmation before deleting them.
4. Run the `UNREGISTER DATABASE` command. For example:

   ```
   UNREGISTER DATABASE;
   ```

Resetting a Database Incarnation in the Recovery Catalog

When you run either the RMAN command or the SQL statement `ALTER DATABASE OPEN RESETLOGS`, you create a new incarnation of the database. You can access a record of the new incarnation in the `V$DATABASE_INCARNATION` view of the target database.

If you run the RMAN command or the SQL statement `ALTER DATABASE OPEN RESETLOGS`, then a new database incarnation record is automatically created in the recovery catalog. The database also implicitly and automatically issues a `RESET DATABASE` command, which specifies that this new incarnation of the database is the current incarnation. All subsequent backups and log archiving done by the target database is associated with the new database incarnation.

In the rare situation in which you wish to restore backups of a prior incarnation of the database, use the `RESET DATABASE TO INCARNATION` *`key`* command to change the current incarnation to an older incarnation. For example, if you accidentally drop a table immediately after the most recent `RESETLOGS`, then you may want to recover the database to just before the time of the most recent `RESETLOGS` and then open it with the `RESETLOGS` option, thereby creating a new incarnation.

To reset the recovery catalog to an older incarnation:

1. Determine the incarnation key of the desired database incarnation. Obtain the incarnation key value by issuing a LIST command:

```
LIST INCARNATION OF DATABASE trgt;

List of Database Incarnations
DB Key  Inc Key   DB Name   DB ID       STATUS     Reset SCN    Reset Time
------- -------   -------   ------      -------    ----------   ----------
1       2         TRGT      1224038686  PARENT     1            02-JUL-02
1       582       TRGT      1224038686  CURRENT    59727        10-JUL-02
```

   The incarnation key is listed in the "Inc Key" column.

2. Reset the database to the old incarnation. For example, enter:

```
RESET DATABASE TO INCARNATION 2;
```

3. If the control file of the previous incarnation is available and mounted, then skip to step 6 of this procedure. Otherwise, shut down the database and start it without mounting. For example:

```
SHUTDOWN IMMEDIATE
STARTUP NOMOUNT
```

4. Restore a control file from the old incarnation. If you have a control file tagged, then specify the tag. Otherwise, you can run the SET UNTIL command, as in this example:

```
RUN
{
  SET UNTIL 'SYSDATE-45';
  RESTORE CONTROLFILE; # only if current control file is not available
}
```

5. Mount the restored control file:

```
ALTER DATABASE MOUNT;
```

6. Run RESTORE and RECOVER commands to restore and recover the database files from the prior incarnation, then open the database with the RESETLOGS option. For example, enter:

```
RESTORE DATABASE;
RECOVER DATABASE;
ALTER DATABASE OPEN RESETLOGS;
```

See Also: *Oracle Database Recovery Manager Reference* for RESET DATABASE syntax, *Oracle Database Recovery Manager Reference* for LIST syntax

Removing Recovery Catalog Records with Status DELETED

Use the `prgrmanc.sql` script to remove recovery catalog records with status DELETED. In releases prior to Oracle9*i*, RMAN updated recovery catalog records to DELETED status after deleting the physical files rather than removing the records.

In Oracle9*i* and later, RMAN always removes catalog records and never updates them to status DELETED. However, records with status DELETED can appear in the recovery catalog when you upgrade a recovery catalog created prior to Oracle9*i* to the current release. For this special case, you can run the `prgrmanc.sql` script.

To remove all backup records with status DELETED:

1. Start a SQL*Plus session and connect to the recovery catalog. This example connects to the database `rcat` as user `rman`:

   ```
   % sqlplus rman/cat@catdb
   ```

2. Run the script `prgrmanc.sql` script, which is stored in an operating system specific location (`$ORACLE_HOME/rdbms/admin` on UNIX):

   ```
   SQL> @prgrmanc
   ```

   The script removes all records with status DELETED from the recovery catalog.

Resynchronizing the Recovery Catalog

When RMAN performs a **resynchronization**, it compares the recovery catalog to either the current control file of the target database or a backup control file and updates the recovery catalog with information that is missing or changed. When resynchronizing, RMAN does the following:

1. Creates a snapshot control file.
2. Compares the recovery catalog to the snapshot control file.
3. Updates the recovery catalog with missing or changed information.

RMAN performs resynchronizations automatically as needed when you execute certain commands, including BACKUP. You can also manually perform a full resynchronization using the RESYNC CATALOG command.

Types of Records That Are Resynchronized

Table 13–2 describes the types of records that RMAN resynchronizes.

Table 13–2 Records Updated during a Resynchronization

Records	Description
Log history	Created when an online redo log switch occurs.
Archived redo logs	Associated with archived logs that were created by archiving an online log, copying an existing archived redo log, or restoring an archived redo log backup set. RMAN tracks this information so that it knows which archived logs it should expect to find.
Backup history	Associated with backup sets, backup pieces, proxy copies, and file copies. The `RESYNC CATALOG` command updates these records when a `BACKUP` command is executed.
Incarnation history	Associated with database incarnations.
Physical schema	Associated with datafiles and tablespaces. If the target database is open, then undo segment information is also updated. Physical schema information in the recovery catalog is updated only when the target has the current control file mounted. If the target database has mounted a backup control file, a freshly created control file, or a control file that is less current than a control file that was seen previously, then physical schema information in the recovery catalog is *not* updated. Physical schema information is also not updated when you use the `RESYNC CATALOG FROM CONTROLFILECOPY` command.

Full and Partial Resynchronization

Resynchronizations can be full or partial. In a partial resynchronization, RMAN reads the current control file to update changed information about new backups, new archived logs, and so forth. However, RMAN does not resynchronize metadata about the database physical schema: datafiles, tablespaces, redo threads, rollback segments (only if the database is open), and online redo logs. In a full

resynchronization, RMAN updates all changed records, including those for the database schema.

Note: Although RMAN performs partial resynchronizations when using a backup control file, it does not perform full resynchronizations. A backup control file may not have correct information about the database physical schema, so a full resynchronization could update the recovery catalog with incorrect information.

Note: You can use Oracle Enterprise Manager to perform catalog resynchronizations.

See Also: *Oracle Database Recovery Manager Reference* for more information about the RESYNC command

When Should You Resynchronize?

RMAN automatically performs full or partial resynchronizations in most situations in which they are needed. Most RMAN commands such as BACKUP, DELETE, and so forth perform a full or partial resynchronization (depending on whether the schema metadata has changed) automatically when the target database control file is mounted and the recovery catalog database is available. Thus, you should not need to manually run RESYNC CATALOG very often.

The following sections describe situations in which a manual resynchronization is required.

Resynchronizing When the Recovery Catalog is Unavailable

If the recovery catalog is unavailable when you issue RMAN commands that cause a partial resynchronization, then open the catalog database later and resynchronize it manually with the RESYNC CATALOG command.

For example, the target database may be in New York while the recovery catalog database is in Japan. You may not want to make daily backups of the target database in CATALOG mode, to avoid depending on the availability of a geographically distant database. In such a case you could connect to the catalog as often as feasible (for example, once each week) and run the RESYNC CATALOG command.

Resynchronizing in ARCHIVELOG Mode When You Back Up Infrequently

Assume that you do the following:

- Run the database in ARCHIVELOG mode
- Back up the database infrequently (for example, hundreds of archive logs are archived between database backups)
- Generate a high number of log switches every day (for example, 1000 switches between catalog resynchronizations)

In this case, you may want to manually resynchronize the recovery catalog regularly because the recovery catalog is *not* updated automatically when a redo log switch occurs or when a redo log is archived. The database stores information about log switches and archived redo logs only in the control file. You must periodically resynchronize in order to propagate this information into the recovery catalog.

How frequently you need to resynchronize the recovery catalog depends on the rate at which the database archives redo logs. The cost of the operation is proportional to the number of records in the control file that have been inserted or changed since the previous resynchronization. If no records have been inserted or changed, then the cost of resynchronization is very low; if many records have been inserted or changed, then the resynchronization is more time-consuming.

Resynchronizing After Physical Database Changes

Resynchronize the recovery catalog after making any change to the physical structure of the target database. As with redo log archive operations, the recovery catalog is *not* updated automatically after physical schema change, including:

- Adding or dropping a tablespace
- Adding a new datafile to an existing tablespace
- Adding or dropping a rollback segment

Forcing a Full Resynchronization of the Recovery Catalog

Use RESYNC CATALOG to force a full resynchronization of the recovery catalog.

1. Connect RMAN to the target and recovery catalog databases, and then mount or open the target database if it is not already mounted or open:

   ```
   STARTUP MOUNT;
   ```

2. Run the RESYNC CATALOG command at the RMAN prompt:

```
RESYNC CATALOG;
```

See Also: *Oracle Database Recovery Manager Reference* for RESYNC CATALOG command syntax

Resynchronizing the Recovery Catalog and CONTROLFILE_RECORD_KEEP_TIME

If you maintain a recovery catalog, then use the RMAN RESYNC CATALOG command often enough to ensure that control file records are propagated to the recovery catalog before they are reused.

Make sure that CONTROLFILE_RECORD_KEEP_TIME is longer than the interval between backups or resynchronizations, or else control file records could be lost. An extra week is a safe margin in most circumstances.

Caution: Never set CONTROL_FILE_RECORD_KEEP_TIME to 0. If you do, then backup records may be overwritten in the control file before RMAN is able to add them to the catalog.

See Also: *Oracle Database Backup and Recovery Basics* to learn how to monitor the overwriting of control file records

Working with RMAN Stored Scripts in the Recovery Catalog

Stored scripts offer an alternative to command files for managing frequently used sequences of RMAN commands.

The chief advantage of a stored script over a command file is that a stored script is always available to any RMAN client that can connect to the target database and recovery catalog, whereas command files are only available if the RMAN client has access to the file system on which they are stored.

Stored scripts can be global or local. A local stored script is associated with the target database to which RMAN is connected when the script is created, and can only be executed when you are connected to that target database. A global stored script can be run against any database registered in the recovery catalog, if the RMAN client is connected to the recovery catalog and a target database.

Note that to work with stored scripts, even global ones, you must be connected to both a recovery catalog and a target instance.

Creating Stored Scripts: CREATE SCRIPT

Make sure RMAN is connected to the right target database and the recovery catalog. Then run the CREATE SCRIPT command, as shown in this example:

```
CREATE SCRIPT full_backup
{
  BACKUP DATABASE PLUS ARCHIVELOG;
  DELETE OBSOLETE;
}
```

Examine the output. If no errors are displayed, then the script was successfully created and stored in the recovery catalog.

For a global script, the syntax is similar:

```
CREATE GLOBAL SCRIPT global_full_backup
{
  BACKUP DATABASE PLUS ARCHIVELOG;
  DELETE OBSOLETE;
}
```

You can also provide a COMMENT with descriptive information:

```
CREATE GLOBAL SCRIPT global_full_backup
COMMENT 'use only with ARCHIVELOG mode databases'
{
  BACKUP DATABASE PLUS ARCHIVELOG;
  DELETE OBSOLETE;
}
```

Finally, you can create a local or global script, reading its contents from a text file:

```
CREATE SCRIPT full_backup FROM FILE 'my_script_file.txt';
```

The file must begin with a { character, contain a series of commands valid within a RUN block, and end with a } character. Otherwise, a syntax error is signalled, just as if the commands were entered at the keyboard.

Running Stored Scripts: EXECUTE SCRIPT

To run a stored script, connect to the target database and recovery catalog, and use EXECUTE SCRIPT. EXECUTE SCRIPT requires a RUN block, as shown:

```
RUN { EXECUTE SCRIPT full_backup; }
```

This command invokes a local script if one is with the name specified. If no local script is found, but there is a global script with the name specified, RMAN will execute the global script. You can also use EXECUTE GLOBAL SCRIPT to control which script is invoked if a local and a global script have the same name. Assuming there is no local script called global_full_backup, the following two commands have the same effect:

```
RUN { EXECUTE GLOBAL SCRIPT global_full_backup; }
RUN { EXECUTE SCRIPT global_full_backup; }
```

Executing a global script only affects the connected target database; to run a global script across multiple databases, you must connect the RMAN client to each one separately and execute the script.

Your script will use the automatic channels configured at the time you execute the script. Use ALLOCATE CHANNEL commands in the script if you need to override the configured channels. Note that, because of the RUN block, if an RMAN command in the script fails, subsequent RMAN commands in the script will not execute.

> **See Also:** *Oracle Database Recovery Manager Reference* for EXECUTE SCRIPT command syntax

Displaying a Stored Script: PRINT SCRIPT

The PRINT SCRIPT command displays a stored script or writes it out to a file. With RMAN connected to the target database and recovery catalog, use the PRINT SCRIPT command as shown here:

```
PRINT SCRIPT full_backup;
```

To send the contents of a script to a file, use this form of the command:

```
PRINT SCRIPT full_backup TO FILE 'my_script_file.txt';
```

For global scripts, the analogous syntax would be:

```
PRINT GLOBAL SCRIPT global_full_backup;
```

and

```
PRINT GLOBAL SCRIPT global_full_backup TO FILE 'my_script_file.txt';
```

> **See Also:** *Oracle Database Recovery Manager Reference* for PRINT SCRIPT command syntax

Listing Stored Scripts: LIST SCRIPT NAMES

Use the LIST SCRIPT NAMES command to display the names of scripts defined in the recovery catalog. This command displays the names of all stored scripts, both global and local, that can be executed for the currently connected target database:

```
LIST SCRIPT NAMES;
```

If RMAN is not connected to a target database when the LIST SCRIPT NAMES command is run, then RMAN will respond with an error.

To view only global script names, use this form of the command:

```
LIST GLOBAL SCRIPT NAMES;
```

To view the names of all scripts stored in the current recovery catalog, including global scripts and local scripts for all target databases registered in the recovery catalog, use this form of the command:

```
LIST ALL SCRIPT NAMES;
```

The output will indicate for each script listed which target database the script is defined for (or whether a script is global).

Note: LIST GLOBAL SCRIPT NAMES and LIST ALL SCRIPT NAMES are the only commands that work when RMAN is connected to a recovery catalog without connecting to a target instance.

See Also: *Oracle Database Recovery Manager Reference* for LIST SCRIPT NAMES command syntax and output format.

Updating Stored Scripts: REPLACE SCRIPT

To update stored scripts, connect to the target database and recovery catalog and use the REPLACE SCRIPT command. If the script does not already exist, then RMAN creates it.

This command updates stored script script full_backup with new contents:

```
REPLACE SCRIPT full_backup
{
  BACKUP DATABASE PLUS ARCHIVELOG;
}
```

Global scripts can be updated using the REPLACE GLOBAL SCRIPT command when connected to a recovery catalog, as follows:

```
REPLACE GLOBAL SCRIPT global_full_backup
COMMENT 'A script for full backup to be used with any database'
{
  BACKUP AS BACKUPSET DATABASE PLUS ARCHIVELOG;
}
```

As with CREATE SCRIPT, you can update a local or global stored script from a text file, with this form of the command:

```
REPLACE GLOBAL SCRIPT global_full_backup FROM FILE 'my_script_file.txt';
```

> **See Also:** *Oracle Database Recovery Manager Reference* for REPLACE SCRIPT command syntax

Deleting Stored Scripts: DELETE SCRIPT

To delete a stored script from the recovery catalog, connect to the catalog and a target database, and use the DELETE SCRIPT command:

```
DELETE SCRIPT 'full_backup';
```

To delete a global stored script, use DELETE GLOBAL SCRIPT:

```
DELETE GLOBAL SCRIPT 'global_full_backup';
```

If you use DELETE SCRIPT without GLOBAL, and there is no stored script for the target database with the specified name, RMAN will look for a global stored script by the specified name and delete the global script if it exists. So, if you were to enter the command

```
DELETE SCRIPT 'global_full_backup';
```

RMAN would look for a script '`global_full_backup`' defined for the connected target database, and if it did not find one, it would search the global scripts for a script called '`global_full_backup`' and delete that script.

> **See Also:** *Oracle Database Recovery Manager Reference* for DELETE SCRIPT command syntax

Starting the RMAN Client and Running a Stored Script

To run the RMAN client and start a stored script in the recovery catalog on startup, use the SCRIPT argument when starting the RMAN client.

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb SCRIPT 'full_backup';
```

You must connect to a recovery catalog (which contains the stored script) and target database (to which the script will apply) when starting the RMAN client.

> **See Also:** *Oracle Database Recovery Manager Reference* for full RMAN client command line syntax

Restrictions on Stored Script Names

There are some issues to be aware of about how RMAN resolves script names, especially when a local and global script share the same name.

- RMAN permits but generally does not require that you use quotes around the name of a stored script. However, if the name begins with a digit or if the name is an RMAN reserved word, you will have to put quotes around the name to use it as a stored script name. Consider avoiding stored script names that begin with characters other than A-Z or that are the same as RMAN reserved words.

  > **See Also:** *Oracle Database Recovery Manager Reference*

- When starting the RMAN client with a SCRIPT argument on the command line, if local and global scripts are defined with the same name, then RMAN will always execute the local script.
- For the EXECUTE SCRIPT, DELETE SCRIPT and PRINT SCRIPT commands, if the script name passed as an argument is not the name of a script defined for the connected target instance, RMAN will look for a global script by the same name to execute, delete or print. For example, if the a stored script global_full_backup is in the recovery catalog as a global script, but no local stored script global_full_backup is defined for the target database, the following command will delete the global script:

  ```
  DELETE SCRIPT global_full_backup;
  ```

Consider using some naming convetion to avoid mistakes due to confusion between global stored scripts and local stored scripts.

> **See Also:** *Oracle Database Recovery Manager Reference* for the list of RMAN reserved words.

Managing the Control File When You Use a Recovery Catalog

Your goal is to ensure that the metadata in the recovery catalog is current. Because the recovery catalog obtains its metadata from the target control file, the currency of the data in the catalog depends on the currency of the data in the control file. You need to make sure that the backup metadata in the control file is recorded in the catalog before it is overwritten with new records.

The `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter determines the minimum number of days that records are retained in the control file before they are candidates for being overwritten. Hence, you must ensure that you resynchronize the recovery catalog with the control file records before these records are erased. As described in "Resynchronizing the Recovery Catalog and CONTROLFILE_RECORD_KEEP_TIME" on page 13-15, you should perform either of the following actions at intervals less than the `CONTROL_FILE_RECORD_KEEP_TIME` setting:

- Make a backup, thereby performing an implicit resynchronization of the recovery catalog
- Manually resynchronize the recovery catalog with the `RESYNC CATALOG` command

So, to ensure the currency of the information in the recovery catalog, the frequency of resynchronizations should be related to the value for the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter.

One problem can arise if the control file becomes too large. The size of the target database control file grows depending on the number of:

- Backups that you perform
- Archived redo logs that the database generates
- Days that this information is stored in the control file

As explained in *Oracle Database Backup and Recovery Basics*, if the control file grows so large that it can no longer expand because it has reached either the maximum number of blocks or the maximum number of records, then the database may overwrite the oldest records even if their age is less than the `CONTROL_FILE_RECORD_KEEP_TIME` setting. In this case, the database writes a message to the alert log. If you discover that this situation occurs frequently, then reducing the value of `CONTROL_FILE_RECORD_KEEP_TIME` and increase the frequency of resynchronizations.

Note: The maximum size of the control file is port-specific. Typically, the maximum size is 20,000 Oracle blocks. Refer to your operating system-specific Oracle documentation for more information.

See Also:

- *Oracle Database Reference* for more information about the `CONTROL_FILE_RECORD_KEEP_TIME` parameter
- *Oracle Database Administrator's Guide* for more detailed information on all aspects of control file management

Backing Up and Recovering the Recovery Catalog

Include the recovery catalog database in your backup and recovery strategy. If you do not back up the recovery catalog and a disk failure occurs that destroys the recovery catalog database, then you may lose the metadata in the catalog.

Backing Up the Recovery Catalog

Here are some general guidelines you should follow when developing a strategy for backing up the recovery catalog.

Back Up the Recovery Catalog Often

The recovery catalog database is a database like any other, and is also a key part of your backup and recovery strategy. Protect the recovery catalog as you would protect any other part of your database, by backing it up. The backup strategy for your recovery catalog database should be part of your overall backup and recovery strategy.

Back up the recovery catalog with the same frequency that you back up the target database. For example, if you make a weekly whole database backup of the target database, then back up the recovery catalog immediately after all target database backups, in order to protect the record of the whole database backup. This backup can help you in a disaster recovery scenario. Even if you have to restore the recovery catalog database using a control file autobackup, you can then use the full record of backups in your restored recovery catalog database to restore the target database without using a control file autobackup for the target database.

Choosing the Appropriate Method for Physical Backups

When backing up the recovery catalog database, you can use RMAN to make the backups. As illustrated in Figure 13–1, you should start RMAN with the NOCATALOG option so that the repository for the recovery catalog is the control file in the catalog database.

Follow these guidelines when developing an RMAN backup strategy for the recovery catalog database:

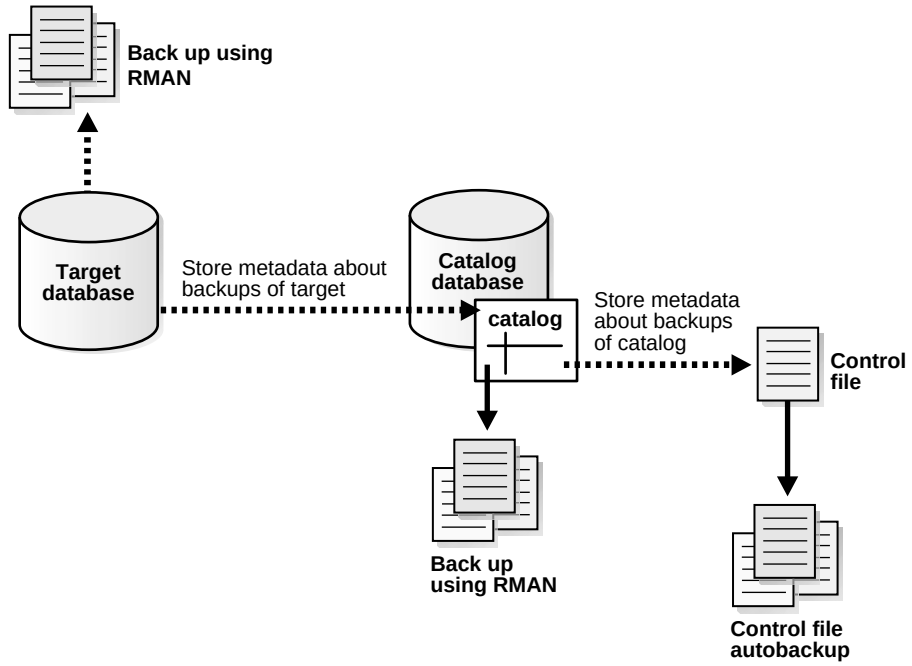
- Run the recovery catalog database in ARCHIVELOG mode so that you can do point-in-time recovery if needed.
- Set the retention policy to a REDUNDANCY value greater than 1.
- Back up the database onto two separate media (for example, disk and tape).
- Run BACKUP DATABASE PLUS ARCHIVELOG at regular intervals, to a media manager if available, or just to disk.
- Do not use another recovery catalog as the repository for the backups.
- Configure the control file autobackup feature to ON.

With this strategy, the control file autobackup feature ensures that the recovery catalog database can always be recovered, so long as the control file autobackup is available.

Exporting the Recovery Catalog Data for Logical Backups

Logical backups of the RMAN recovery catalog created with one of Oracle's export utilities can be a useful supplement for physical backups. In the event of damage to the recovery catalog database, you can quickly reimport the exported recovery catalog data into another database and rebuild the catalog this way.

Figure 13–1 Using the Control File as the Repository for Backups of the Catalog



See Also: "Performing Disaster Recovery" on page 8-18 for more information for recovery with a control file autobackup

Storing the Recovery Catalog in an Appropriate Place

Never store a recovery catalog containing the RMAN repository for a database in the same database as the target database or on the same disks as the target database. For example, do not store the catalog for database `prod1` in `prod1`. A recovery catalog for `prod1` is only effective if it is separated from the data that it is designed to protect.

If `prod1` suffers a total media failure, and if the recovery catalog data for `prod1` is also stored in `prod1`, then you have no catalog to aid in recovery. You will probably have to restore an autobackup of the control file for `prod1` and use it to restore and recover the database.

Separating the target and catalog databases is especially important when you back up a recovery catalog database. The following example shows what you should *not* do. For example, consider the following:

- Target database `prod1` and catalog database `catdb` are on different hosts.
- `catdb` contains the recovery catalog repository for target database `prod1`.

You decide to use a recovery catalog to back up `catdb`, but are not sure where to create it. If you create the catalog containing the repository for `catdb` in database `catdb`, then if you lose `catdb` due to a media failure, you will have difficulty restoring `catdb` *and* will leave `prod1` without a recovery catalog to use in a restore scenario.

Restoring and Recovering the Recovery Catalog from Backup

Restoring and recovering the recovery catalog is much like restoring and recovering any other database, if you backed it up with RMAN.

You can restore the control file and SPFILE for the recovery catalog database from an autobackup, then restore and perform complete recovery on the rest of the database. The processes required are all described in *Oracle Database Backup and Recovery Basics* You can also use another recovery catalog to record metadata about backups of this recovery catalog database, if you are in a situation where you are using multiple recovery catalogs.

Re-Creating the Recovery Catalog

If the recovery catalog database is lost or damaged, and recovery of the recovery catalog database through the normal Oracle recovery procedures is not possible, then you must re-create the catalog. Examples of this worst-case scenario include:

- A recovery catalog database that has never been backed up
- A recovery catalog database that has been backed up, but cannot be recovered because the datafile backups or archived logs are not available

You have these options for partially re-creating the contents of the missing recovery catalog:

- Issue `CATALOG START WITH...` commands to recatalog backups.
- Use the `RESYNC CATALOG` command to extract metadata from a control file and rebuild the recovery catalog. Note that you automatically lose any metadata that was contained in old control file records that aged out of the control file.

Depending on the state of the target control file, you can:

- Resynchronize from the current control file
- Resynchronize from a control file copy

See Also:

- *Oracle Database Recovery Manager Reference* for information about the CATALOG command
- *Oracle Database Recovery Manager Reference* for information about the CROSSCHECK command
- "Managing the Control File When You Use a Recovery Catalog" on page 13-21 to learn about how records age out of the control file

Exporting and Importing the Recovery Catalog

To move the recovery catalog from one database to another, export the catalog from the old database, and import it into the new one. You can only import the catalog into a supported version of the Oracle database server. In general, you can import the catalog into a database of the same release or later.

Exports can also serve as logical backups of the RMAN recovery catalog. If the recovery catalog database is damaged, you can quickly reimport the exported recovery catalog data into another database and rebuild the catalog.

This section contains the following topics:

Considerations When Moving Catalog Data

You should only import the recover catalog into a schema that does not already contain a recovery catalog schema. In other words, the user who will own the imported recovery catalog schema should not already own a recovery catalog schema. For example, if user `rman` owns the recovery catalog on database `catdb`, and you want to export the recovery catalog on `catdb` and import it into database `catdb2`, then `rman` should not already own a recovery catalog on `catdb2`. You should either create a new recovery catalog owner on `catdb2`, or drop the current

user rman on catdb2 and then re-create the user. You cannot merge a recovery catalog into an existing recovery catalog.

The basic steps for exporting a recovery catalog from a primary database and importing the catalog into a secondary database are as follows:

1. Use one of the Oracle Export utilities to export the catalog data from the primary database. See "Exporting the Recovery Catalog" on page 13-27 for an example.
2. Create a user on the secondary database as described in "Creating the Recovery Catalog Owner" on page 13-3, and grant the user necessary privileges.
3. Use the import utility corresponding to the export utility in step 1 to import the catalog data into the schema created in the previous step. See "Importing the Recovery Catalog" on page 13-28 for an example.

You should *not* run the CREATE CATALOG command either before or after the Import of the catalog into the secondary database. By importing the catalog data into the new schema, you effectively create the catalog in the secondary database.

> **Note:** You cannot import data exported from two different recovery catalogs to merge them into one catalog. It is not currently possible to merge two or more recovery catalog schemas into one.

Exporting the Recovery Catalog

This example uses the Original Export utility described in *Oracle Database Utilities* to create a logical export of the recovery catalog. Refer to *Oracle Database Utilities* for concepts and procedures relating to the Data Pump Export utility.

The following procedure creates a logical export of the recovery catalog.

1. Execute the Oracle export utility at the operating system command line, making sure to do the following:
   a. Connect as the owner of the recovery catalog
   b. Specify the OWNER option
   c. Specify an output file

   For example, if the owner of the catalog in database catdb is rman, you can issue the following at the UNIX command line to export the catalog to file cat.dmp:

   ```
   % exp rman/cat@catdb FILE=cat.dmp OWNER=rman
   ```

2. Examine the output to make sure you were successful:

```
Export terminated successfully without warnings.
```

Importing the Recovery Catalog

This example uses the Original Import utility described in *Oracle Database Utilities* to create a logical export of the recovery catalog. Refer to *Oracle Database Utilities* for concepts and procedures relating to the Data Pump Import utility.

To make a logical import of the recovery catalog from the command line:

1. Create a new user in another database. For the recommended SQL syntax for creating a new user in a recovery catalog database, see "Creating the Recovery Catalog Owner" on page 13-3. Be sure to grant the new user the necessary privileges.
2. Import the catalog data from the export file. Execute the import at the command line, making sure to do the following:
   a. Connect as the new owner of the recovery catalog.
   b. Specify the old owner with the `FROMUSER` parameter.
   c. Specify the new owner with the `TOUSER` parameter.
   d. Specify the import file.

   For example, assume the following:

   - The old owner of the catalog in database `prod1` is `rman`.
   - The user in the new recovery catalog database `catdb2` is `rman2`.
   - The file containing the export of the catalog is `cat.dmp`.

   The command is then as follows:

```
% imp USERID=rman2/cat2@catdb2 FILE=cat.dmp FROMUSER=rman TOUSER=rman2
```

3. Use the imported catalog data for restore and recovery of your target database.

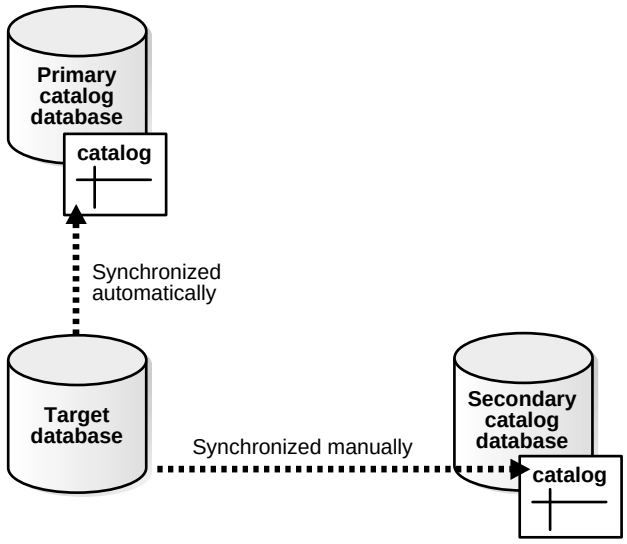
Increasing Availability of the Recovery Catalog

You may have a production system in which you want to maintain high availability for the catalog database. For example, you may have 100 target databases registered in the recovery catalog. In case the primary catalog database goes down, you can

create redundancy by storing a secondary recovery catalog in a separate database, as illustrated in Figure 13–2. You must register the target database in the secondary catalog.

In this availability scenario, the main catalog is synchronized as normal during regular backups, while the secondary catalog is synchronized periodically with the RESYNC CATALOG command. If the primary catalog database goes down or requires routine maintenance, then you can resynchronize the secondary catalog and use it as the new primary catalog during the interim.

Figure 13–2 Registering One Target Database in Two Recovery Catalogs



See Also: "Creating a Recovery Catalog" on page 13-2

Querying the Recovery Catalog Views

The LIST, REPORT, and SHOW commands should provide you with all the repository information that you require. Nevertheless, you can sometimes also obtain useful information from the recovery catalog views, which are views in the catalog schema prefixed with RC_.

See Also: *Oracle Database Recovery Manager Reference* for reference information about the recovery catalog views

RMAN obtains backup and recovery metadata from the target database control file and stores it in the catalog tables. The recovery catalog views are derived from these tables. Note that these views are not normalized or optimized for user queries.

In general, the recovery catalog views are not as user-friendly as the RMAN reporting commands. For example, when you start RMAN and connect to a target database, you obtain the information for this target database only when you issue `LIST`, `REPORT`, and `SHOW` commands. If you have 10 different target databases registered in the same recovery catalog, then the catalog views show the information for all incarnations of all databases registered in the catalog. You often have to perform joins among the views to distinguish the specific incarnation of the target database that you are interested in.

Most of the catalog views have a corresponding dynamic performance view (or V$ view) in the database server. For example, `RC_BACKUP_PIECE` corresponds to `V$BACKUP_PIECE`. The primary difference between the recovery catalog view and corresponding server view is that each catalog view contains information about all the databases registered in the catalog, whereas the database server view contains information only about itself. The RC_ views and corresponding V$ views use different primary keys to uniquely identify rows.

Distinguishing a Database in the Catalog Views

Most of the catalog views contain the columns `DB_KEY` and `DBINC_KEY`. Each target database can be uniquely identified by either the primary key, which is the `DB_KEY` column value, or the `DBID`, which is the 32-bit unique database identifier. Each incarnation of each target database is uniquely identified by the `DBINC_KEY` primary key. When querying data about a specific incarnation of a target database, you should use these columns to specify the database. Then, you can perform joins with most of the other catalog views to obtain the desired information.

Distinguishing a Database Object in the Catalog Views

An important difference between catalog and V$ views is that a different system of unique identifiers is used for backup and recovery objects. For example, many `V$` views such as `V$ARCHIVED_LOG` use the `RECID` and `STAMP` columns to form a concatenated primary key. The corresponding catalog view uses a derived value as its primary keys and stores this value in a single column. For example, the primary key in `RC_ARCHIVED_LOG` is the `AL_KEY` column. The `AL_KEY` column value is the primary key that RMAN displays in the `LIST` command output.

Querying Catalog Views for the Target DB_KEY or DBID Values

The DB_KEY value, which is the primary key for a target database, is used only in the recovery catalog. The easiest way is to obtain the DB_KEY is to use the DBID of the target database, which is displayed whenever you connect RMAN to the target database. The DBID, which is a unique system-defined number given to every Oracle database, is what distinguishes one target database from another target database in the RMAN metadata.

Assume that you want to obtain information about one of the target databases registered in the recovery catalog. You can easily determine the DBID from this database either by looking at the output displayed when RMAN connects to the database, querying V$RMAN_OUTPUT, or querying a V$DATABASE view as in the following:

```
SELECT DBID
FROM V$DATABASE;

DBID
---------
598368217
```

You can then obtain the DB_KEY for a target database by running the following query, where *dbid_of_target* is the DBID that you previously obtained:

```
SELECT DB_KEY
FROM RC_DATABASE
WHERE DBID = dbid_of_target;
```

To obtain information about the current incarnation of a target database, specify the target database DB_KEY value and perform a join with RC_DATABASE_INCARNATION by using a WHERE condition to specify that the CURRENT_INCARNATION column value is set to YES. For example, to obtain information about backup sets in the current incarnation of a target database with the DB_KEY value of 1, you can execute the following script:

```
SELECT BS_KEY, BACKUP_TYPE, COMPLETION_TIME
  FROM RC_DATABASE_INCARNATION i, RC_BACKUP_SET b
  WHERE i.DB_KEY = 1
  AND i.DB_KEY = b.DB_KEY
  AND i.CURRENT_INCARNATION = 'YES';
```

You should use the DB_NAME column to specify a database *only if* you do not have more than one database registered in the recovery catalog with the same DB_NAME. RMAN permits you to register more than one database with the same database

name, but requires that the DBID values be different. For example, you can have ten databases with the DB_NAME value of `prod1`, each with a different DBID. Because the DBID is the unique identifier for every database in the metadata, use this value to obtain the DB_KEY and then use DB_KEY to uniquely identify the database.

Determining the Schema Version of the Recovery Catalog

The schema version of the recovery catalog is stored in the recovery catalog itself. The information is important in case you maintain multiple databases of different versions in your production system, and need to determine whether the catalog schema version is usable with a specific target database version.

> **See Also:** *Oracle Database Recovery Manager Reference* for the complete set of compatibility rules governing the RMAN environment

To determine the schema version of the recovery catalog:

1. Start SQL*Plus and connect to the recovery catalog database as the catalog owner. For example:

   ```
   % sqlplus rman/cat@catdb
   ```

2. Query the RCVER table to obtain the schema version, as in the following example (sample output included):

   ```
   SELECT *
   FROM rcver;

   VERSION
   ------------
   09.02.00
   ```

If the table displays multiple rows, then the highest version in the RCVER table is the current catalog schema version. The table stores only the major version numbers and not the patch numbers. For example, assume that the `rcver` table displays the following rows:

```
VERSION
------------
08.01.07
09.00.01
09.02.00
```

These rows indicate that the catalog was created with a release 8.1.7 executable, then upgraded to release 9.0.1, and finally upgraded to release 9.2.0. The current version of the catalog schema is 9.2.0.

Upgrading the Recovery Catalog

If you use a version of the recovery catalog that is older than that required by the RMAN client, then you must upgrade it. For example, you must upgrade the catalog if you use a release 8.1 version of the RMAN client with a release 8.0 version of the recovery catalog.

You receive an error when issuing UPGRADE CATALOG if the recovery catalog is already at a version greater than that required by the RMAN client. RMAN permits the UPGRADE CATALOG command to be run if the recovery catalog is current and does not require upgrading, however, so that you can re-create packages at any time if necessary. Check the message log for error messages generated during the upgrade.

To upgrade the recovery catalog:

1. To install the new recovery catalog schema, the recovery catalog user must have TYPE privilege:

   ```
   sqlplus> connect sys/oracle@catdb as sysdba;
   sqlplus> grant TYPE to rman;
   ```

2. Use RMAN to connect to the target and recovery catalog databases. For example, enter:

   ```
   % rman TARGET / CATALOG rman/cat@catdb

   connected to recovery catalog database
   PL/SQL package rcat.DBMS_RCVCAT version 08.00.04 in RCVCAT database
   is too old
   ```

3. Issue the UPGRADE CATALOG command:

   ```
   UPGRADE CATALOG;

   recovery catalog owner is rman
   enter UPGRADE CATALOG command again to confirm catalog upgrade
   ```

4. Enter the UPDATE CATALOG command again to confirm:

   ```
   UPGRADE CATALOG;
   ```

```
recovery catalog upgraded to version 09.02.00
DBMS_RCVMAN package upgraded to version 09.02.00
DBMS_RCVCAT package upgraded to version 09.02.00
```

See Also:

- *Oracle Database Recovery Manager Reference* for UPGRADE CATALOG command syntax
- *Oracle Database Recovery Manager Reference* for information about recovery catalog compatibility
- *Oracle Database Upgrade Guide* for complete compatibility and migration information

Dropping the Recovery Catalog

If you do not want to maintain a recovery catalog, then you can drop the recovery catalog schema from the tablespace. The DROP CATALOG command deletes all information from the recovery catalog. Hence, if you have no backups of the recovery catalog schema, then backups of all target databases managed by this catalog may become unusable. (The control file of the target database will still retain a record of recent backups.)

The DROP CATALOG command is not appropriate for unregistering a single database from a recovery catalog that has multiple target databases registered. Dropping the catalog deletes the recovery catalog record of backups for all target databases registered in the catalog.

To drop the recovery catalog schema:

1. Use RMAN to connect to the target and recovery catalog databases.

   ```
   % rman TARGET / CATALOG rman/cat@catdb
   ```

2. Issue the DROP CATALOG command twice to confirm:

   ```
   DROP CATALOG;

   recovery catalog owner is rman
   enter DROP CATALOG command again to confirm catalog removal

   DROP CATALOG;
   ```

Note: Even after you drop the recovery catalog, the control file still contains records about the backups. To purge RMAN repository records from the control file, re-create the control file.

See Also: *Oracle Database Recovery Manager Reference* for DROP CATALOG command syntax, and "Unregistering a Target Database from the Recovery Catalog" on page 13-8 to learn how to unregister a database from the catalog

14

Tuning Backup and Recovery

Tuning RMAN performance is mostly a matter of maximizing the speed with which RMAN creates your backups and restores from backups, on disk and especially on tape. A secondary concern is limiting the effect of backup activities on database throughput.

You may also need to tune performance of the database during instance recovery.

This chapter covers the concepts needed for performance tuning, and the features in RMAN that can help you. The discussion is divided into the following sections:

Tuning Recovery Manager: Overview

RMAN backup and restore operations have the following distinct components:

- Reading or writing input data
- Processing data by validating blocks and copying them from the input to the output buffers

The slowest of these operations in any RMAN task is called the **bottleneck**. RMAN tuning involves identifying the bottlenecks for a given task and using RMAN commands, initialization parameter settings, or adjustments to physical media to improve performance on the backup.

The key to tuning RMAN is understanding how it performs I/O. RMAN's backup and restore jobs use two types of I/O buffers: `DISK` and tertiary storage (usually tape). When performing a backup, RMAN reads input files using disk buffers and writes the output backup file by using either disk or tape buffers. Restore operations use disk or tape buffers for input, depending on where the backup is stored, and disk buffers for output.

To tune RMAN effectively, you must thoroughly understand concepts such as synchronous and asynchronous I/O, disk and tape buffers, and channel architecture. When you understand these concepts, then you can learn how to use fixed views to monitor bottlenecks, and use the techniques described in "Tuning RMAN Backup Performance: Examples" on page 14-8 to solve problems.

There are a number of concepts that affect RMAN performance and that can therefore influence your strategy for backup performance tuning:

- I/O Buffer Allocation
- Allocation of Tape Buffers
- Synchronous and Asynchronous I/O
- Factors Affecting Backup Speed to Tape
- Using the RATE Parameter to Control Disk Bandwidth Usage

I/O Buffer Allocation

RMAN I/O uses two different types of buffers: disk and tape. These buffers are typically different sizes. They are allocated differently, depending upon the device type and the role the buffer plays in an RMAN operation.

Allocation for Disk Buffers

To understand how RMAN allocates buffers to read datafiles during backups, you must understand how RMAN multiplexing works.

RMAN **multiplexing** is RMAN's ability to read a number of files in a backup simultaneously from different sources to improve reading performance, and then write them to a single backup piece. The level of multiplexing is the number of files read simultaneously.

Multiplexing is described at greater length in "Multiplexed Backup Sets" on page 2-16. The level of multiplexing is determined by the algorithm described in "Algorithm for Multiplexed Backups" on page 2-17. Review this section before proceeding.

When RMAN backs up from disk, it uses the datafile described in the following table to determine how large to make the buffers.

Table 14–1 Datafile Read Buffer Sizing Algorithm

Level of Multiplexing	Resulting Buffer Size
Less than or equal to 4	RMAN allocates buffers of size 1 MB so that the total buffer size for all the input files is 16 MB.
Greater than 4 but less than or equal to 8	RMAN allocates disk buffers of size 512 KB so that the total buffer size for all the files is less than 16 MB.
Greater than 8	RMAN allocates a fixed 4 disk buffers of 128 KB for each file, so that the total size is 512 KB for each file.

The number of buffers allocated depends on the following rules:

- When the ouput of the backup resides on disk, 4 buffers are allocated, their size being operating system dependent.
- If the operation is a restore, and the backup resides on disk, 4 buffers are allocated, their size being operating system dependent.
- When restoring a backup, for each active datafile 4 buffers of 128K are allocated.
- When image copies are produced, only 4 buffers in total are allocated, each of an operating system dependent size.

Allocation of Tape Buffers

If you backup to or restore from an `sbt` device, then by default the database allocates four buffers for each channel for the tape writers (or reads if doing a

restore). The size of these buffers is platform dependent, but is typically 256K. This value can be changed using the ALLOCATE or SEND command using the PARMS and the BLKSIZE option.

To calculate the total size of buffers used during a backup or restore, multiply the buffer size by 2, and then multiply this product by the number of channels.

For example, assume that you use two tape channels and each buffer is 256K. In this case, the total size of buffers used during a backup is as follows:

```
256KB/buffer x 4 buffers/channel x 2 channels = 2 MB
```

RMAN allocates the tape buffers in the SGA if I/O slaves are being used, or the PGA otherwise.

If you use I/O slaves, then set the LARGE_POOL_SIZE initialization parameter to set aside SGA memory dedicated to holding these large memory allocations. This prevents RMAN I/O buffers from competing with the library cache for SGA memory. If I/O slaves for tape I/O were requested but there is not enough space in the SGA for them, slaves are not used, and a message appears in the alert log.

Synchronous and Asynchronous I/O

When RMAN reads or writes data, the I/O is either **synchronous** or **asynchronous**. When the I/O is synchronous, a server process can perform only one task at a time. When it is asynchronous, a server process can begin an I/O and then perform other work while waiting for the I/O to complete. It can also begin multiple I/O operations before waiting for the first to complete.

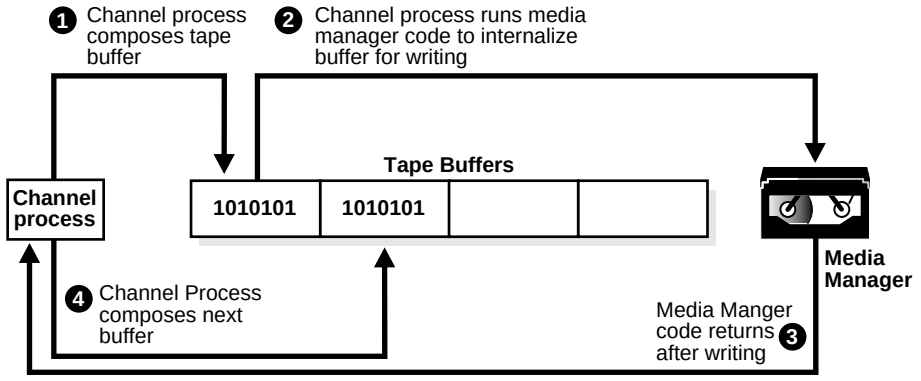
Some operating systems support native asynchronous disk I/O. The database takes advantage of this feature if it is available. On operating systems that do not support native asynchronous I/O, the database can simulate it with special I/O slave processes that are dedicated to performing I/O on behalf of another process. You can control disk I/O slaves by setting the DBWR_IO_SLAVES parameter to a nonzero value. The database allocates four backup disk I/O slaves for any nonzero value of DBWR_IO_SLAVES.

By contrast, tape I/O is always synchronous. For tape I/O, each channel allocated (whether manually or based on a CONFIGURE command) corresponds to a server process, called here a channel process.

Synchronous I/O: Example

Figure 14–1 shows synchronous I/O in a backup to tape.

Figure 14–1 Synchronous I/O



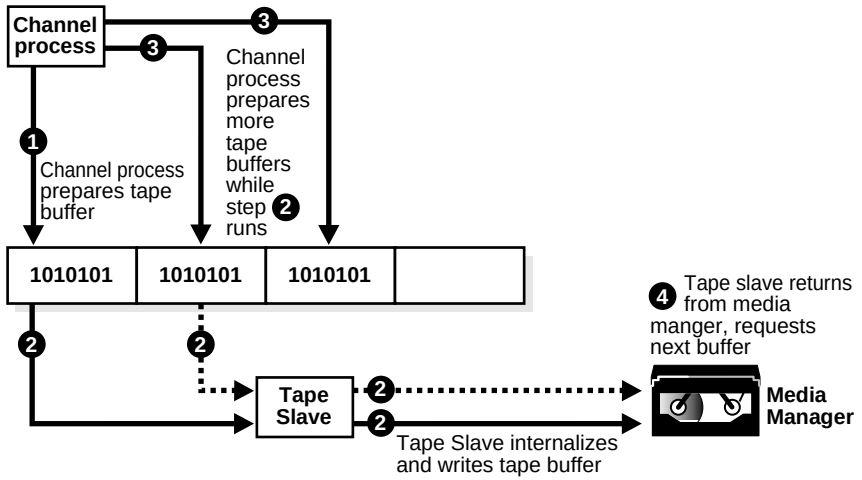
The following steps occur:

1. The channel process composes a tape buffer.
2. The channel process executes media manager code that processes the tape buffer and internalizes it for further processing and storage by the media manager.
3. The media manager code returns a message to the server process stating that it has completed writing.
4. The channel process can initiate a new task.

Figure 14–2 shows **asynchronous I/O** in a tape backup. Asynchronous I/O to tape is simulated by using tape slaves. In this case, each allocated channel corresponds to a server process, which in the explanation which follows is identified as a channel

process. For each channel process, one tape slave is started (or more than one, in the case of multiple copies).

Figure 14–2 Asynchronous I/O



The following steps occur:

1. A channel process writes blocks to a tape buffer.
2. The channel process sends a message to the tape slave process to process the tape buffer. The tape slave process executes media manager code that processes the tape buffer and internalizes it so that the media manager can process it.
3. While the tape slave process is writing, the channel process is free to read data from the datafiles and prepare more output buffers.
4. Once the tape slave channel returns from the media manager code, it requests a new tape buffer, which usually is ready. Thus waiting time for the channel process is reduced, and the backup is completed faster.

Factors Affecting Backup Speed to Tape

The following factors affect the speed of the backup to tape:

- Native Transfer Rate
- Tape Compression
- Tape Streaming

- Physical Tape Block Size

Native Transfer Rate

The tape native transfer rate is the speed of writing to a tape without compression. This speed represents the upper limit of the backup rate. The upper limit of your backup performance should be the aggregate transfer rate of all of your tape drives. If your backup is already performing at that rate, and if it is not using an excessive amount of CPU, then RMAN performance tuning will not help.

Tape Compression

The level of tape compression is very important for backup performance. If the tape has good compression, then the sustained backup rate is faster. For example, if the compression ratio is 2:1 and native transfer rate of the tape drive is 6 MB/s, then the resulting backup speed is 12 MB/s. In this case, RMAN must be able to read disks with a throughput of more than 12 MB/s or the disk becomes the bottleneck for the backup.

Note: You should not use both tape compression provided by the media manager and binary backupset compression as provided by RMAN. If the media manager compression is efficient, then it is usually the better choice. Using RMAN compressed backupsets can be an effective alternative if you need to reduce bandwidth used to move uncompressed backupsets over a network to the media manager, and if the CPU overhead required to compress the data in RMAN is acceptable.

See *Oracle Database Backup and Recovery Basics* for more on using compressed backupsets.

Tape Streaming

Tape **streaming** during write operations has a major impact on tape backup performance. Almost all tape drives currently on the market are fixed-speed, streaming tape drives. Because such drives can only write data at one speed, when they run out of data to write to tape, the tape must slow down and stop. Generally, when the drive's buffer empties, the tape is moving so quickly that it actually overshoots; to continue writing, the drive must rewind the tape to locate the point where it stopped writing.

Physical Tape Block Size

The physical tape block size can affect backup performance. The block size is the amount of data written by media management software to a tape in one write operation. In general, the larger the tape block size, the faster the backup. Note that physical tape block size is not controlled by RMAN or the Oracle database server, but by media management software. See your media management software's documentation for details.

Features and Options Used to Tune RMAN Performance

There are a number of features you can use to tune your backup performance, once you have sufficient knowledge of your database and its workload and bottlenecks.

Using the RATE Parameter to Control Disk Bandwidth Usage

The RATE parameter specifies the bytes/second that RMAN reads on this channel. Use this parameter to set an upper limit for bytes read so that RMAN does not consume excessive disk bandwidth and degrade online performance.

For example, set `RATE=1500K`. If each disk drive delivers 3 MB/second, then RMAN leaves some disk bandwidth available to the online system.

Tuning RMAN Backup Performance: Examples

Many factors can affect backup performance. Often, finding the solution to a slow backup is a process of trial and error. To get the best performance for a backup, follow the suggested steps in this section:

- Step 1: Remove RATE Parameters from Configured and Allocated Channels
- Step 2: If You Use Synchronous Disk I/O, Set DBWR_IO_SLAVES
- Step 3: If You Fail to Allocate Shared Memory, Set LARGE_POOL_SIZE
- Step 4: Determine Whether Files Are Empty or Contain Few Changes
- Step 5: Query V$ Views to Identify Bottlenecks

Step 1: Remove RATE Parameters from Configured and Allocated Channels

The RATE parameter on a channel is intended to reduce, rather than increase, backup throughput, so that more disk bandwidth is available for other database operations.

If your backup is not streaming to tape, then make sure that the RATE parameter is not set on the ALLOCATE CHANNEL or CONFIGURE CHANNEL commands.

Step 2: If You Use Synchronous Disk I/O, Set DBWR_IO_SLAVES

If and only if your disk does *not* support asynchronous I/O, then try setting the DBWR_IO_SLAVES initialization parameter to a nonzero value. Any nonzero value for DBWR_IO_SLAVES causes a fixed number (four) of disk I/O slaves to be used for backup and restore, which simulates asynchronous I/O. If I/O slaves are used, I/O buffers are obtained from the SGA. The large pool is used, if configured. Otherwise, the shared pool is used.

Note: By setting DBWR_IO_SLAVES, the database writer processes will use slaves as well. You may need to increase the value of the PROCESSES initialization parameter.

Step 3: If You Fail to Allocate Shared Memory, Set LARGE_POOL_SIZE

Set this initialization parameter if the database reports an error in the alert.log stating that it does not have enough memory and that it will not start I/O slaves. The message looks something like the following:

```
ksfqxcre: failure to allocate shared memory means sync I/O will be used whenever
async I/O to file not supported natively
```

When attempting to get shared buffers for I/O slaves, the database does the following:

- If LARGE_POOL_SIZE is set, then the database attempts to get memory from the large pool. If this value is not large enough, then an error is recorded in the alert log, the database does not try to get buffers from the shared pool, and asynchronous I/O is not used.
- If LARGE_POOL_SIZE is not set, then the database attempts to get memory from the shared pool.
- If the database cannot get enough memory, then it obtains I/O buffer memory from the PGA and writes a message to the alert.log file indicating that synchronous I/O is used for this backup.

The memory from the large pool is used for many features, including the shared server (formerly called multi-threaded server), parallel query, and RMAN I/O slave buffers. Configuring the large pool prevents RMAN from competing with other subsystems for the same memory.

Requests for contiguous memory allocations from the shared pool are usually small (under 5 KB) in size. However, it is possible that a request for a large contiguous memory allocation can either fail or require significant memory housekeeping to release the required amount of contiguous memory. Although the shared pool may be unable to satisfy this memory request, the large pool is able to do so. The large pool does not have a least recently used (LRU) list; the database does not attempt to age memory out of the large pool.

Use the LARGE_POOL_SIZE initialization parameter to configure the large pool. To see in which pool (shared pool or large pool) the memory for an object resides, query V$SGASTAT.POOL.

The formula for setting LARGE_POOL_SIZE is as follows:

```
LARGE_POOL_SIZE =  number_of_allocated_channels *
                   (16 MB + ( 4 *  size_of_tape_buffer ) )
```

> **See Also:** *Oracle Database Concepts* for more information about the large pool, and *Oracle Database Reference* for complete information about initialization parameters

Step 4: Determine Whether Files Are Empty or Contain Few Changes

When performing a full backup of files that are largely empty, or when performing an incremental backup when block change tracking is disabled and few blocks have changed, RMAN may not be able to supply blocks with data to the tape fast enough to keep it streaming. In either case, you can improve performance by increasing the level of multiplexing.

> **Note:** One reliable way to determine whether the tape streaming or disk I/O is the bottleneck in a given backup job is to compare the time required to run backup tasks with the time required to run BACKUP VALIDATE of the same tasks. BACKUP VALIDATE of a backup to tape performs the same disk reads as a real backup but performs no tape I/O. If the time required for the BACKUP VALIDATE to tape is significantly less than the time required for a real backup to tape, then writing to tape is the likely bottleneck.

An incremental backup is an RMAN backup in which only modified blocks are backed up. If change tracking is disabled, then incremental backups are often slow because the database must read the entire datafile to find blocks which have

changed. If tape drives are not locally attached, then incremental backups can be faster. You must consider how much bandwidth exists for reading the disks compared to the bandwidth for writing to the tapes. If tape bandwidth is limited compared to disk, then incremental backups may help.

If only a few blocks have changed in an incremental backup, then you need to input many buffers from the datafile before you accumulate enough blocks to fill a buffer and write to tape. Hence, the tape drive may not stream.

Step 5: Query V$ Views to Identify Bottlenecks

If none of the previous steps improves backup performance, then try to determine the exact source of the bottleneck. Use the V$BACKUP_SYNC_IO and V$BACKUP_ASYNC_IO views to determine the source of backup or restore bottlenecks and to see detailed progress of backup jobs.

V$BACKUP_SYNC_IO contains rows when the I/O is synchronous to the process (or thread on some platforms) performing the backup. V$BACKUP_ASYNC_IO contains rows when the I/O is asynchronous. Asynchronous I/O is obtained either with I/O processes or because it is supported by the underlying operating system.

> **See Also:** *Oracle Database Reference* for more information about these views

To determine whether your tape is streaming when the I/O is synchronous, query the EFFECTIVE_BYTES_PER_SECOND column in the V$BACKUP_SYNC_IO or V$BACKUP_ASYNC_IO view. If EFFECTIVE_BYTES_PER_SECOND is less than the raw capacity of the hardware, then the tape is not streaming. If EFFECTIVE_BYTES_PER_SECOND is greater than the raw capacity of the hardware, the tape may or may not be streaming. Compression may cause the EFFECTIVE_BYTES_PER_SECOND to be greater than the speed of real I/O.

Identifying Bottlenecks with Synchronous I/O

With synchronous I/O, it is difficult to identify specific bottlenecks because all synchronous I/O is a bottleneck to the process. The only way to tune synchronous I/O is to compare the rate (in bytes/second) with the device's maximum throughput rate. If the rate is lower than the rate that the device specifies, then consider tuning this aspect of the backup and restore process. The DISCRETE_BYTES_PER_SECOND column in the V$BACKUP_SYNC_IO view displays the I/O rate. If you see data in V$BACKUP_SYNC_IO, then the problem is that you have not enabled asynchronous I/O or you are not using disk I/O slaves.

Identifying Bottlenecks with Asynchronous I/O

Long waits are the number of times the backup or restore process told the operating system to wait until an I/O was complete. **Short waits** are the number of times the backup or restore process made an operating system call to poll for I/O completion in a nonblocking mode. **Ready** indicates the number of time when I/O was already ready for use and so there was no need to made an operating system call to poll for I/O completion.

The simplest way to identify the bottleneck is to query V$BACKUP_ASYNC_IO for the datafile that has the largest ratio for LONG_WAITS divided by IO_COUNT.

Note: If you have synchronous I/O but you have set BACKUP_DISK_IO_SLAVES, then the I/O will be displayed in V$BACKUP_ASYNC_IO.

See Also: *Oracle Database Reference* for descriptions of the V$BACKUP_SYNC_IO and V$BACKUP_ASYNC_IO views

Instance Recovery Performance Tuning: FAST_START_MTTR_TARGET

This section offers guidelines for tuning the time required to perform crash and instance recovery. It contains the following topics:

- Understanding Instance Recovery
- Checkpointing and Cache Recovery
- Configuring the Duration of Cache Recovery: FAST_START_MTTR_TARGET
- Tuning FAST_START_MTTR_TARGET and Using MTTR Advisor

Understanding Instance Recovery

Instance and crash recovery are the automatic application of redo log records to Oracle data blocks after a crash or system failure. During normal operation, if an instance is shut down cleanly (as when using a SHUTDOWN IMMEDIATE statement), rather than terminated abnormally, then the in-memory changes that have not already been written to the datafiles on disk are written to disk as part of the checkpoint performed during shutdown.

However, if a single instance database crashes or if all instances of an Oracle Real Application Cluster configuration crash, then Oracle performs crash recovery at the

next startup. If one or more instances of an Oracle Real Application Cluster configuration crash, then a surviving instance performs instance recovery automatically. Instance and crash recovery occur in two steps: cache recovery followed by transaction recovery.

The database can be opened as soon as cache recovery completes, so improving the performance of cache recovery is important for increasing availability.

Cache Recovery (Rolling Forward)

During the cache recovery step, Oracle applies all committed and uncommitted changes in the redo log files to the affected data blocks. The work required for cache recovery processing is proportional to the rate of change to the database (update transactions each second) and the time between checkpoints.

Transaction Recovery (Rolling Back)

To make the database consistent, the changes that were not committed at the time of the crash must be undone (in other words, rolled back). During the transaction recovery step, Oracle applies the rollback segments to undo the uncommitted changes.

Checkpointing and Cache Recovery

Periodically, Oracle records a checkpoint. A checkpoint is the highest system change number (SCN) such that all data blocks less than or equal to that SCN are known to be written out to the data files. If a failure occurs, then only the redo records containing changes at SCNs higher than the checkpoint need to be applied during recovery. The duration of cache recovery processing is determined by two factors: the number of data blocks that have changes at SCNs higher than the SCN of the checkpoint, and the number of log blocks that need to be read to find those changes.

How Checkpoints Affect Performance

Frequent checkpointing writes dirty buffers to the datafiles more often than otherwise, and so reduces cache recovery time in the event of an instance failure. If checkpointing is frequent, then applying the redo records in the redo log between the current checkpoint position and the end of the log involves processing relatively few data blocks. This means that the cache recovery phase of recovery is fairly short.

However, in a high-update system, frequent checkpointing can reduce runtime performance, because checkpointing causes DBWn processes to perform writes.

Fast Cache Recovery Trade-offs To minimize the duration of cache recovery, you must force Oracle to checkpoint often, thus keeping the number of redo log records to be applied during recovery to a minimum. However, in a high-update system, frequent checkpointing increases the overhead for normal database operations.

If daily operational efficiency is more important than minimizing recovery time, then decrease the frequency of writes to data files due to checkpoints. This should improve operational efficiency, but also increase cache recovery time.

Configuring the Duration of Cache Recovery: FAST_START_MTTR_TARGET

The fast-start fault recovery feature reduces the time required for cache recovery and makes the recovery bounded and predictable by limiting the number of dirty buffers and the number of redo records generated between the most recent redo record and the last checkpoint.

The foundation of fast-start fault recovery is the fast-start checkpointing architecture. Instead of conventional event-driven (that is, log switching) checkpointing, which does bulk writes, fast-start checkpointing occurs incrementally. Each DBWn process periodically writes buffers to disk to advance the checkpoint position. The oldest modified blocks are written first to ensure that every write lets the checkpoint advance. Fast-start checkpointing eliminates bulk writes and the resultant I/O spikes that occur with conventional checkpointing.

With the fast-start fault recovery feature,the FAST_START_MTTR_TARGET initialization parameter simplifies the configuration of recovery time from instance or system failure. FAST_START_MTTR_TARGET specifies a target for the expected mean time to recover (MTTR), that is, the time (in seconds) that it should take to start up the instance and perform cache recovery. Once FAST_START_MTTR_TARGET is set, the database manages incremental checkpoint writes in an attempt to meet that target. If you have chosen a practical value for FAST_START_MTTR_TARGET, you can expect your database to recover, on average, in approximately the number of seconds you have chosen.

Note: You must disable or remove the FAST_START_IO_TARGET, LOG_CHECKPOINT_INTERVAL, and LOG_CHECKPOINT_TIMEOUT initialization parameters when using FAST_START_MTTR_TARGET. Setting these parameters interferes with the mechanisms used to manage cache recovery time to meet FAST_START_MTTR_TARGET.

Practical Values for FAST_START_MTTR_TARGET The maximum value for FAST_START_MTTR_TARGET is 3600 seconds (one hour). If you set the value to more than 3600, then Oracle rounds it to 3600.

The following example shows how to set the value of FAST_START_MTTR_TARGET:

```
SQL> ALTER DATABASE SET FAST_START_MTTR_TARGET=30;
```

In principle, the minimum value for FAST_START_MTTR_TARGET is one second. However, the fact that you can set FAST_START_MTTR_TARGET this low does not mean that that target can be achieved. There are practical limits to the minimum achievable MTTR target, due to such factors as database startup time.

The MTTR target that your database can achieve given the current value of FAST_START_MTTR_TARGET is called the **effective MTTR target**. You can view your current effective MTTR by viewing the TARGET_MTTR column of the V$INSTANCE_RECOVERY view.

The practical range of MTTR target values for your database is defined to be the range between the lowest achieveable effective MTTR target for your database and the longest that startup and cache recovery will take in the worst-case scenario (that is, when the whole buffer cache is dirty). A procedure for determining the range of achievable MTTR target values, one step in the process of tuning your FAST_START_MTTR_TARGET value, is described in "Determine the Practical Range for FAST_START_MTTR_TARGET" on page 14-17.

> **Note:** It is usually not useful to set your FAST_START_MTTR_TARGET to a value outside the practical range. If your FAST_START_MTTR_TARGET value is shorter than the lower limit of the practical range, the effect is as if you set it to the lower limit of the practical range. In such a case, the effective MTTR target will be the best MTTR target the system can achieve, but checkpointing will be at a maximum, which can affect normal database performance. If you set FAST_START_MTTR_TARGET to a time longer than the practical range, the MTTR target will be no better than the worst-case situation.

Reducing Checkpoint Frequency to Optimize Runtime Performance To reduce the checkpoint frequency and optimize runtime performance, you can do the following:

- Set the value of FAST_START_MTTR_TARGET to 3600. This enables fast-start checkpointing and the fast-start fault recovery feature, but minimizes its effect

on runtime performance while avoiding the need for performance tuning of FAST_START_MTTR_TARGET.

- Size your online redo log files according to the amount of redo your system generates. A good rule of thumb is to switch logs at most every twenty minutes. Having your log files too small can increase checkpoint activity and reduce performance. Also note that all redo log files should be the same size.

  See Also: *Oracle Database Concepts* for a complete discussion of checkpoints

Monitoring Cache Recovery with V$INSTANCE_RECOVERY The V$INSTANCE_RECOVERY view displays the current recovery parameter settings. You can also use statistics from this view to determine which factor has the greatest influence on checkpointing.

The following table lists those columns most useful in monitoring predicted cache recovery performance:

Table 14–2 V$INSTANCE_RECOVERY Columns

Column	Description
TARGET_MTTR	Effective mean time to recover (MTTR) target in seconds. This field is 0 if FAST_START_MTTR_TARGET is not specified.
ESTIMATED_MTTR	The current estimated mean time to recover (MTTR) in seconds, based on the current number of dirty buffers and log blocks. This field is always calculated, whether or not FAST_START_MTTR_TARGET is specified.

For more details on the columns in V$INSTANCE_RECOVERY, see *Oracle Database Reference*.

As part of the ongoing monitoring of your database, you can periodically compare V$INSTANCE_RECOVERY.TARGET_MTTR to your FAST_START_MTTR_TARGET. The two values should generally be the same if the FAST_START_MTTR_TARGET value is in the practical range. If TARGET_MTTR is consistently longer than FAST_START_MTTR_TARGET, then set FAST_START_MTTR_TARGET to a value no less than TARGET_MTTR. If TARGET_MTTR is consistently shorter, then set FAST_START_MTTR_TARGET to a value no greater than TARGET_MTTR.

Tuning FAST_START_MTTR_TARGET and Using MTTR Advisor

To determine the appropriate value for FAST_START_MTTR_TARGET for your database, execute the following four step process:

- Calibrate the FAST_START_MTTR_TARGET
- Determine the Practical Range for FAST_START_MTTR_TARGET
- Evaluate Different Target Values with MTTR Advisor
- Determine Optimal Size for Redo Logs

Calibrate the FAST_START_MTTR_TARGET

The FAST_START_MTTR_TARGET initialization parameter causes the database to calculate internal system trigger values, in order to limit the length of the redo log and the number of dirty data buffers in the data cache. This calculation uses estimated time to read a redo block, estimates of the time to read and write a data block, as well as characteristics of typical workload of the system, such as how many dirty buffers corresponds to how many change vectors, and so on.

Initially, internal defaults are used in the calculation. These defaults are replaced over time by data gathered on I/O performance during system operation and actual cache recoveries.

You will have to perform several instance recoveries in order to calibrate your FAST_START_MTTR_TARGET value properly. Before starting calibration, you must decide whether FAST_START_MTTR_TARGET is being calibrated for a database crash or a hardware crash. This is a consideration if your database files are stored in a file system or if your I/O subsystem has a memory cache, because there is a considerable difference in the read and write time to disk depending on whether or not the files are cached. The appropriate value for FAST_START_MTTR_TARGET will depend upon which type of crash is more important to recover from quickly.

To effecitvely calibrate FAST_START_MTTR_TARGET, make sure that you run the typcial workload of the system for long enough, and perform several instance recoveries to ensure that the time to read a redo block and the time to read or write a data block during recovery are recorded accurately.

Determine the Practical Range for FAST_START_MTTR_TARGET

After calibration, you can perform tests to determine the practical range for FAST_START_MTTR_TARGET for your database.

Determining Lower Bound for FAST_START_MTTR_TARGET: Scenario To determine the lower bound of the practical range, set FAST_START_MTTR_TARGET to 1, and start up your database. Then check the value of V$INSTANCE_RECOVERY.TARGET_MTTR, and use this value as a good lower bound for FAST_START_MTTR_TARGET. Database startup time, rather than cache recovery time, is usually the dominant factor in determining this limit.

For example, set the FAST_START_MTTR_TARGET to 1:

```
SQL> ALTER DATABASE SET FAST_START_MTTR_TARGET=1;
```

Then, execute the following query immediately after opening the database:

```
SQL> SELECT TARGET_MTTR, ESTIMATED_MTTR
    FROM V$INSTANCE_RECOVERY;
```

Oracle responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
18          15
```

The TARGET_MTTR value of 18 seconds is the minimum MTTR target that the system can achieve, that is, the lowest practical value for FAST_START_MTTR_TARGET. This minimum is calculated based on the average database startup time.

The ESTIMATED_MTTR field contains the estimated mean time to recovery based on the current state of the running database. Because the database has just opened, the system contains few dirty buffers, so not much cache recovery would be required if the instance failed at this moment. That is why ESTIMATED_MTTR can, for the moment, be lower than the minimum possible TARGET_MTTR.

ESTIMATED_MTTR can be affected in the short term by recent database activity. Assume that you query V$INSTANCE_RECOVERY immediately after a period of heavy update activity in the database. Oracle responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
18          30
```

Now the effective MTTR target is still 18 seconds, and the estimated MTTR (if a crash happened at that moment) is 30 seconds. This is an acceptable result. This means that some checkpoints writes might not have finished yet, so the buffer cache contains more dirty buffers than targeted.

Now wait for sixty seconds and reissue the query to V$INSTANCE_RECOVERY. Oracle responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
```

```
18          25
```

The estimated MTTR at this time has dropped to 25 seconds, because some of the dirty buffers have been written out during this period

Determining Upper Bound for FAST_START_MTTR_TARGET To determine the upper bound of the practical range, set FAST_START_MTTR_TARGET to 3600, and operate your database under a typical workload for a while. Then check the value of V$INSTANCE_RECOVERY.TARGET_MTTR. This value is a good upper bound for FAST_START_MTTR_TARGET.

The procedure is substantially similar to that in "Determining Lower Bound for FAST_START_MTTR_TARGET: Scenario" on page 14-18.

Selecting Preliminary Value for FAST_START_MTTR_TARGET Once you have determined the practical bounds for the FAST_START_MTTR_TARGET parameter, select a preliminary value for the parameter. Choose a higher value within the practical range if your concern is with database performance, and a lower value within the practical range if your priority is shorter recovery times. The narrower the practical range, of course, the easier the choice becomes.

For example, if you discovered that the practical range was between 17 and 19 seconds, it would be quite simple to choose 19, because it makes relatively little difference in recovery time and at the same time minimizes the effect of checkpointing on system performance. However, if you found that the practical range was between 18 and 40 seconds, you might choose a compromise value of 30, and set the parameter accordingly:

```
SQL> ALTER DATABASE SET FAST_START_MTTR_TARGET=30;
```

You might then go on to use the MTTR Advisor to determine an optimal value.

Evaluate Different Target Values with MTTR Advisor

Once you have selected a preliminary value for FAST_START_MTTR_TARGET, you can use MTTR Advisor to evaluate the effect of different FAST_START_MTTR_TARGET settings on system performance, compared to your chosen setting.

Enabling MTTR Advisor To enable MTTR Advisor, set the two initialization parameters STATISTICS_LEVEL and FAST_START_MTTR_TARGET.

STATISTICS_LEVEL governs whether all advisors are enabled and is not specific to MTTR Advisor. Make sure that it is set to TYPICAL or ALL. Then, when FAST_START_MTTR_TARGET is set to a non-zero value, the MTTR Advisor is enabled.

Using MTTR Advisor After enabling MTTR Advisor, run a typical database workload for a while. When MTTR Advisor is ON, the database simulates checkpoint queue behavior under the current value of FAST_START_MTTR_TARGET, and up to four other different MTTR settings within the range of valid FAST_START_MTTR_TARGET values. (The database will in this case determine the valid range for FAST_START_MTTR_TARGET itself before testing different values in the range.)

Viewing MTTR Advisor Results: V$MTTR_TARGET_ADVICE The dynamic performance view V$MTTR_TARGET_ADVICE lets you view statistics or advisories collected by MTTR Advisor.

The database populates V$MTTR_TARGET_ADVICE with advice about the effects of each of the FAST_START_MTTR_TARGET settings for your database. For each possible value of FAST_START_MTTR_TARGET, the row contains details about how many cache writes would be performed under the workload tested for that value of FAST_START_MTTR_TARGET.

Specifically, each row contains information about cache writes, total phyiscal writes (including direct writes), and total I/O (including reads) for that value of FAST_START_MTTR_TARGET, expressed both as a total number of operations and a ratio compared to the operations under your chosen FAST_START_MTTR_TARGET value. For instance, a ratio of 1.2 indicates 20% more cache writes.

Knowing the effect of different FAST_START_MTTR_TARGET settings on cache write activity and other I/O enables you to decide better which FAST_START_MTTR_TARGET value best fits your recovery and performance needs.

If MTTR Advisor is currently on,V$MTTR_TARGET_ADVICE shows the Advisor information collected. If MTTR Advisor is currently OFF, the view shows information collected the last time MTTR Advisor was ON since database startup, if any. If the database has been restarted since the last time the MTTR Advisor was used, or if it has never been used, the view will not show any rows.

> **See Also:** *Oracle Database Reference* for column details of the V$MTTR_TARGET_ADVICE view

Determine Optimal Size for Redo Logs

You can use the V$INSTANCE_RECOVERY view column OPTIMAL_LOGFILE_SIZE to determine the size of your online redo logs. This field shows the redo log file size in megabytes that is considered optimal based on the current setting of FAST_START_MTTR_TARGET. If this field consistently shows a value greater than the size of your smallest online log, then you should configure all your online logs to be at least this size.

Note, however, that the redo log file size affects the MTTR. In some cases, you may be able to refine your choice of the optimal FAST_START_MTTR_TARGET value by re-running the MTTR Advisor with your suggested optimal log file size.

15

Recovery Manager Troubleshooting

This chapter describes how to troubleshoot Recovery Manager. This chapter contains these topics:

- Interpreting RMAN Message Output
- Testing the Media Management API
- Terminating an RMAN Command
- RMAN Troubleshooting Scenarios

Interpreting RMAN Message Output

Recovery Manager provides detailed error messages that can aid in troubleshooting problems. Also, the Oracle database server and third-party media vendors generate useful debugging output of their own. The discussion which follows explains how to identify and interpret the different errors you may encounter.

Identifying Types of Message Output

Output that is useful for troubleshooting failed or hung RMAN jobs is located in several different places, as explained in the following table.

Type of Output	Produced By	Location	Description
RMAN messages	RMAN	Completed job information is in V$RMAN_STATUS and RC_RMAN_STATUS. Current job information is in V$RMAN_OUTPUT. When running RMAN from the command line, you can direct output to the following places: ■ Standard output ■ A log file specified by LOG on the command line or the SPOOL LOG command ■ A file created by redirecting RMAN output (for example, UNIX > operator)	Contains actions relevant to the RMAN job as well as error messages generated by RMAN, the database server, and the media vendor. RMAN error messages have an RMAN-*xxxxx* prefix. Normal action descriptions do not have a prefix.
alert_*SID*.log	Oracle database server	The directory named in the BACKGROUND_DUMP_DEST initialization parameter.	Contains a chronological log of errors, initialization parameter settings, and administration operations. Records values for overwritten control file records (refer to *Oracle Data Guard Concepts and Administration*).
Oracle trace file	Oracle database server	The directory specified in the USER_DUMP_DEST initialization parameter.	Contains detailed output generated by Oracle server processes. This file is created when an ORA-600 or ORA-3113 error message occurs, whenever RMAN cannot allocate a channel, and when the database fails to load the media management library.

Type of Output	Produced By	Location	Description
`sbtio.log`	Third-party media management software	The directory specified in the `USER_DUMP_DEST` initialization parameter.	Contains vendor-specific information written by the media management software. This log does not contain Oracle server or RMAN errors.
Media manager log file	Third-party media management software	The filenames for any media manager logs other than `sbtio.log` are determined by the media management software.	Contains information on the functioning of the media management device.

Recognizing RMAN Error Message Stacks

RMAN reports errors as they occur. If an error is not retriable, that is, RMAN cannot perform failover to another channel to complete a particular job step, then RMAN also reports a summary of the errors after all job sets complete. This feature is known as **deferred error reporting**.

One way to determine whether RMAN encountered an error is to examine its return code, as described in "Identifying RMAN Return Codes" on page 15-10. A second way is to search the RMAN output for the string `RMAN-00569`, which is the message number for the error stack banner. All RMAN errors are preceded by this error message. If you do not see an `RMAN-00569` message in the output, then there are no errors. Following is sample output for a syntax error:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-00558: error encountered while parsing input commands
RMAN-01005: syntax error: found ")": expecting one of: "archivelog, backup,
backupset, controlfilecopy, current, database, datafile, datafilecopy, (, plus,
;, tablespace"
RMAN-01007: at line 1 column 18 file: standard input
```

Identifying Error Codes

Typically, you find the following types of error codes in RMAN message stacks:

- Errors prefixed with `RMAN-`
- Errors prefixed with `ORA-`
- Errors preceded by the line `Additional information:`

See Also: *Oracle Database Error Messages* for explanations of RMAN and ORA error codes

RMAN Error Message Numbers

Table 15–1 indicates the error ranges for common RMAN error messages, all of which are described in *Oracle Database Error Messages*.

Table 15–1 RMAN Error Message Ranges

Error Range	Cause
0550-0999	Command-line interpreter
1000-1999	Keyword analyzer
2000-2999	Syntax analyzer
3000-3999	Main layer
4000-4999	Services layer
5000-5499	Compilation of RESTORE or RECOVER command
5500-5999	Compilation of DUPLICATE command
6000-6999	General compilation
7000-7999	General execution
8000-8999	PL/SQL programs
9000-9999	Low-level keyword analyzer
10000-10999	Server-side execution
11000-11999	Interphase errors between PL/SQL and RMAN
12000-12999	Recovery catalog packages

ORA-19511: Media Manager Errors

In the event of a media manager error, ORA-19511 is signalled, and the media manager is expected to provide RMAN a descriptive error. RMAN will display the error passed back to it by the media manager. For example, you might see this:

```
ORA-19511: Error received from media manager layer, error text:
   sbtpvt_open_input: file .* does not exist or cannot be accessed, errno = 2
```

The message from the media manager should provide you with enough information to let you fix the root problem. If it does not, you should refer to the

documentation for your media manager or contact your media management vendor support representative for further information. `ORA-19511` errors originate with the media manager, not the Oracle database. The database merely passes the message on from the media manager. The cause can only be addressed by the media management vendor.

Note that if you are still using an SBT 1.1-compliant media management layer, you may see some additional error message text. Output from an SBT 1.1-compliant media management layer is similar to the following:

```
ORA-19507: failed to retrieve sequential file, handle="c-140148591-20031014-06",
parms=""
ORA-27007: failed to open file
Additional information: 7000
Additional information: 2
ORA-19511: Error received from media manager layer, error text:
   SBT error = 7000, errno = 0, sbtopen: backup file not found
```

The "Additional information" provided uses error codes specific to SBT 1.1. The values displayed correspond to the media manager message numbers and error text listed in Table 15–2. RMAN re-signals the error, as an ORA-19511 `Error received from media manager layer` error, and a general error message related to the error code returned from the media manager and including the SBT 1.1 error number is then displayed.

The SBT 1.1 error messages are listed here for your reference. Table 15–2 lists media manager message numbers and their corresponding error text. In the error codes, *`O/S`* stands for *operating system*. The errors prefixed with an asterisk are internal and should not typically be seen during normal operation.

Table 15–2 Media Manager Error Message Ranges **(Page 1 of 2)**

Cause	No.	Message
sbtopen	7000	Backup file not found (only returned for read)
	7001	File exists (only returned for write)
	7002*	Bad mode specified
	7003	Invalid block size specified
	7004	No tape device found
	7005	Device found, but busy; try again later
	7006	Tape volume not found
	7007	Tape volume is in-use
	7008	I/O Error
	7009	Can't connect with Media Manager
	7010	Permission denied
	7011	O/S error for example malloc, fork error
	7012*	Invalid argument(s) to sbtopen
sbtclose	7020*	Invalid file handle or file not open
	7021*	Invalid flags to sbtclose
	7022	I/O error
	7023	O/S error
	7024*	Invalid argument(s) to sbtclose
	7025	Can't connect with Media Manager
sbtwrite	7040*	Invalid file handle or file not open
	7041	End of volume reached
	7042	I/O error
	7043	O/S error
	7044*	Invalid argument(s) to sbtwrite
sbtread	7060*	Invalid file handle or file not open
	7061	EOF encountered
	7062	End of volume reached
	7063	I/O error
	7064	O/S error
	7065*	Invalid argument(s) to sbtread

Table 15–2 Media Manager Error Message Ranges **(Page 2 of 2)**

Cause	No.	Message
sbtremove	7080	Backup file not found
	7081	Backup file in use
	7082	I/O Error
	7083	Can't connect with Media Manager
	7084	Permission denied
	7085	O/S error
	7086*	Invalid argument(s) to sbtremove
sbtinfo	7090	Backup file not found
	7091	I/O Error
	7092	Can't connect with Media Manager
	7093	Permission denied
	7094	O/S error
	7095*	Invalid argument(s) to sbtinfo
sbtinit	7110*	Invalid argument(s) to sbtinit
	7111	O/S error

Interpreting RMAN Error Stacks

Sometimes you may find it difficult to identify the useful messages in the RMAN error stack. Note the following tips and suggestions:

- Read the messages from the bottom up, because this is the order in which RMAN issues the messages. The last one or two errors displayed in the stack are often the most informative.
- When using an SBT 1.1 media management layer and presented with SBT 1.1 style error messages containing the "`Additional information:`" numeric error codes, look for the `ORA-19511` message that follows for the text of error messages passed back to RMAN by the media manager. These should identify the real failure in the media management layer.
- Look for the `RMAN-03002` or `RMAN-03009` message (`RMAN-03009` is the same as `RMAN-03002` but includes the channel ID), immediately following the error banner. These messages indicate which command failed. Syntax errors generate `RMAN-00558`.

- Identify the basic type of error according to the error range chart in Table 15–1 and then refer to *Oracle Database Error Messages* for information on the most important messages.

Interpreting RMAN Errors: Example

You attempt a backup of tablespace `users` and receive the following message:

```
Starting backup at 29-AUG-02
using channel ORA_DISK_1
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of backup command at 08/29/2002 15:14:03
RMAN-20202: tablespace not found in the recovery catalog
RMAN-06019: could not translate tablespace name "USESR"
```

The `RMAN-03002` error indicates that the `BACKUP` command failed. You read the last two messages in the stack first and immediately see the problem: no tablespace `usesr` appears in the recovery catalog because you mistyped the name.

Interpreting Server Errors: Example

Assume that you attempt to recover a tablespace and receive the following errors:

```
RMAN> RECOVER TABLESPACE users;

Starting recover at 29-AUG-01
using channel ORA_DISK_1

starting media recovery
media recovery failed
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of recover command at 08/29/2001 15:18:43
RMAN-11003: failure during parse/execution of SQL statement: alter database
recover if needed tablespace USERS
ORA-00283: recovery session canceled due to errors
ORA-01124: cannot recover data file 8 - file is in use or recovery
ORA-01110: data file 8: '/oracle/oradata/trgt/users01.dbf'
```

As suggested, you start reading from the bottom up. The `ORA-01110` message explains there was a problem with the recovery of datafile `users01.dbf`. The second error indicates that the database cannot recover the datafile because it is in

use or already being recovered. The remaining RMAN errors indicate that the recovery session was cancelled due to the server errors. Hence, you conclude that because you were not already recovering this datafile, the problem must be that the datafile is online and you need to take it offline and restore a backup.

Interpreting SBT 2.0 Media Management Errors: Example

Assume that you use a tape drive and receive the following output during a backup job:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
ORA-19624: operation failed, retry possible
ORA-19507: failed to retrieve sequential file, handle="/tmp/foo", parms=""
ORA-27029: skgfrtrv: sbtrestore returned error
ORA-19511: Error received from media manager layer, error text:
  sbtpvt_open_input:file /tmp/foo does not exist or cannot be accessed, errno=2
```

The error text displayed following the `ORA-19511` error is generated by the media manager and describes the real source of the failure. Refer to the media manager documentation to interpret this error.

Interpreting SBT 1.1 Media Management Errors: Example

Assume that you use a tape drive and receive the following output during a backup job:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of backup command on c1 channel at 09/04/2001 13:18:19
ORA-19506: failed to create sequential file, name="07d36ecp_1_1", parms=""
ORA-27007: failed to open file
SVR4 Error: 2: No such file or directory
Additional information: 7005
Additional information: 1
ORA-19511: Error received from media manager layer, error text:
   SBT error = 7005, errno = 2, sbtopen: system error
```

The main information of interest returned by SBT 1.1 media managers is the error code in the "Additional information" line:

```
Additional information: 7005
```

Referring to Table 15–2, "Media Manager Error Message Ranges", you discover that error `7005` means that the media management device is busy. So, the media management software is not able to write to the device because it is in use or there is a problem with it.

Note: The `sbtio.log` contains information written by the media management software, *not* the Oracle database server. Hence, you must consult your media vendor documentation to interpret the error codes and messages. If no information is written to the `sbtio.log`, contact your media manager support to ask whether they are writing error messages in some other location, or whether there are steps you need to take to have the media manager errors appear in sbtio.log.

Identifying RMAN Return Codes

One way to determine whether RMAN encountered an error is to examine its return code or exit status. The RMAN client returns 0 to the shell from which it was invoked if no errors occurred, and a nonzero error value otherwise.

How you access this return code depends upon the environment from which you invoked the RMAN client. For example, if you are running UNIX with the C shell, then, when RMAN completes, the return code is placed in a shell variable called `$status`. The method of returning exit status is a detail specific to the host operating system rather than the RMAN client.

Testing the Media Management API

On some platforms, Oracle provides a diagnostic tool called `sbttest`. This utility performs a simple test of the media management software by attempting to communicate with the media manager as the Oracle database server would.

Obtaining the sbttest Utility

On UNIX, the `sbttest` utility is typically located in `$ORACLE_HOME/bin`. If for some reason the utility is not included with your platform, then contact Oracle Support to obtain the C version of the program. You can compile this version of the program on all UNIX platforms.

Note that on platforms such as Solaris, you do not have to relink when using `sbttest`. On other platforms, relinking may be necessary.

Obtaining Online Documentation for the sbttest Utility

For online documentation of `sbttest`, issue the following on the command line:

```
% sbttest
```

The program displays the list of possible arguments for the program:

```
Error: backup file name must be specified
Usage: sbttest backup_file_name # this is the only required parameter
               <-dbname database_name>
               <-trace trace_file_name>
               <-remove_before>
               <-no_remove_after>
               <-read_only>
               <-no_regular_backup_restore>
               <-no_proxy_backup>
               <-no_proxy_restore>
               <-file_type n>
               <-copy_number n>
               <-media_pool n>
               <-os_res_size n>
               <-pl_res_size n>
               <-block_size block_size>
               <-block_count block_count>
               <-proxy_file os_file_name bk_file_name
                           [os_res_size pl_res_size block_size block_count]>
               <-libname sbt_library_name>
```

The display also indicates the meaning of each argument. For example, following is the description for two optional parameters:

```
Optional parameters:
  -dbname  specifies the database name which will be used by SBT
           to identify the backup file. The default is "sbtdb"
  -trace   specifies the name of a file where the Media Management
           software will write diagnostic messages.
```

Using the sbttest Utility

Use `sbttest` to perform a quick test of the media manager. The following table explains how to interpret the output.

If sbttest returns . . .	Then . . .
0	The program ran without error. In other words, the media manager is installed and can accept a data stream and return the same data when requested.
a nonzero value	The program encountered an error. Either the media manager is not installed or it is not configured correctly.

To use sbttest:

1. Make sure the program is installed and included in the system path by typing `sbttest` at the command line:

   ```
   % sbttest
   ```

   If the program is operational, then you should see a display of the online documentation.

2. Execute the program, specifying any of the arguments described in the online documentation. For example, enter the following to create test file `some_file.f` and write the output to `sbtio.log`:

   ```
   % sbttest some_file.f -trace sbtio.log
   ```

   You can also test a backup of an existing datafile. For example, this command tests datafile `tbs_33.f` of database `prod`:

   ```
   % sbttest tbs_33.f -dbname prod
   ```

3. Examine the output. If the program encounters an error, then it provides messages describing the failure. For example, if the database cannot find the library, you see:

   ```
   libobk.so could not be loaded. Check that it is installed properly, and that
   LD_LIBRARY_PATH environment variable (or its equivalent on your platform)
   includes the directory where this file can be found. Here is some additional
   information on the cause of this error:
   ld.so.1: sbttest: fatal: libobk.so: open failed: No such file or directory
   ```

Note that in some cases `sbttest` can work but an RMAN backup does not. The reasons can be the following:

- The user who starts `sbttest` is not the owner of the Oracle processes.

- If the database server is not linked with the media management library or cannot load it dynamically when needed, then RMAN backups to the media manager fail, but `sbttest` may still work.
- The `sbttest` program passes all environment parameters from the shell but RMAN does not.

Terminating an RMAN Command

There are several ways to terminate an RMAN command in the middle of execution:

- The preferred method is to press `CTRL+C` (or the equivalent "attention" key combination for your system) in the RMAN interface. This will also terminates allocated channels, unless they are hung in the media management code, as happens when, for example, when they are waiting for a tape to be mounted.
- You can kill the server session corresponding to the RMAN channel by running the SQL `ALTER SYSTEM KILL SESSION` statement.
- You can terminate the server session corresponding to the RMAN channel on the operating system.

Terminating the Session with ALTER SYSTEM KILL SESSION

You can identify the Oracle session ID for an RMAN channel by looking in the RMAN log for messages with the format shown in the following example:

```
channel ch1: sid=15 devtype=SBT_TAPE
```

The `sid` and `devtype` are displayed for each allocated channel. Note that the Oracle `sid` is different from the operating system process ID. You can kill the session using a SQL `ALTER SYSTEM KILL SESSION` statement.

`ALTER SYSTEM KILL SESSION` takes two arguments, the `sid` printed in the RMAN message and a serial number, both of which can be obtained by querying `V$SESSION`. For example, run the following statement, where *`sid_in_rman_output`* is the number from the RMAN message:

```
SELECT SERIAL# FROM V$SESSION WHERE SID=sid_in_rman_output;
```

Then, run the following statement, substituting the *`sid_in_rman_output`* and serial number obtained from the query:

```
ALTER SYSTEM KILL SESSION 'sid_in_rman_output,serial#';
```

Note that this will not unhang the session if the session is hung in media manager code..

Terminating the Session at the Operating System Level

Finding and killing the processes that are associated with the server sessions is operating system specific. On some platforms the server sessions are not associated with any processes at all. Refer to your operating system specific documentation for more information.

Terminating an RMAN Session That Is Hung in the Media Manager

You may sometimes need to kill an RMAN job that is hung in the media manager. The best way to terminate RMAN when the channel connections are hung in the media manager is to kill the session in the media manager. If this action does not solve the problem, then on some platforms, such as Unix, you may be able to kill the Oracle processes of the connections. (Note that killing the Oracle processes may cause problems from the media manager. See your media manager documentation for details.)

Components of an RMAN Session

The nature of an RMAN session depends on the operating system. In UNIX, an RMAN session has the following processes associated with it:

- The **RMAN client process** itself
- The **default channel**, the initial connection to the target database
- One **target connection** to the target database corresponding to each allocated channel
- The **catalog connection** to the recovery catalog database, if you use a recovery catalog
- An **auxiliary connection** to an auxiliary instance, during `DUPLICATE` or TSPITR operations
- A **polling connection** to the target database, used for monitoring RMAN command execution on the various allocated channels. By default, RMAN makes one polling connection. RMAN makes additional polling connections if you use different connect strings in the `ALLOCATE CHANNEL` or `CONFIGURE CHANNEL` commands. One polling connection exists for each distinct connect string used in the `ALLOCATE CHANNEL` or `CONFIGURE CHANNEL` command.

Process Behavior During a Hung Job

RMAN usually hangs because one of the channel connections is waiting in the media manager code for a tape resource. The catalog connection and the default channel appear to hang, because they are waiting for RMAN to tell them what to do. Polling connections seem to be in an infinite loop while polling the RPC under the control of the RMAN process.

If you kill the RMAN process itself, then you also kill the catalog connection, the auxiliary connection, the default channel, and the polling connections. If target and auxiliary connections are not hung in the media manager code, they also terminate. If either the target connection or any of the auxiliary connections are executing in the media management layer, they will not terminate until the processes are manually killed at the operating system level.

Not all media managers can detect the termination of the Oracle process. Those which cannot may keep resources busy or continue processing. Consult your media manager documentation for details.

Terminating the catalog connection does not cause the RMAN process to terminate because RMAN is not performing catalog operations while the backup or restore is in progress. Removing default channel and polling connections causes the RMAN process to detect that one of the channels has died and then proceed to exit. In this case, the connections to the hung channels remain active as described previously.

Terminating an RMAN Session: Basic Steps

Once the hung channels in the media manager code are killed, the RMAN process detects this termination and proceed to exit, removing all connections except target connections that are still operative in the media management layer. The caveat about the media manager resources still applies in this case.

To terminate an Oracle process that is hung in the media manager:

1. Query V$SESSION and V$SESSION_WAIT as described in "Monitoring RMAN Through V$ Views" on page 4-9. For example, execute the following query:

```
COLUMN EVENT FORMAT a10
COLUMN SECONDS_IN_WAIT FORMAT 999
COLUMN STATE FORMAT a20
COLUMN CLIENT_INFO FORMAT a30

SELECT p.SPID, EVENT, SECONDS_IN_WAIT AS SEC_WAIT,
       STATE, CLIENT_INFO
FROM V$SESSION_WAIT sw, V$SESSION s, V$PROCESS p
WHERE sw.EVENT LIKE 'sbt%'
```

```
       AND s.SID=sw.SID
       AND s.PADDR=p.ADDR
;
```

Examine the SQL output to determine which sbt functions are waiting. For example, the output may be as follows:

```
SPID EVENT        SEC_WAIT STATE               CLIENT_INFO
---- ---------- ---------- ------------------- -------------
8642 sbtwrite2         600 WAITING             rman channel=ORA_SBT_TAPE_1
8374 sbtwrite2         600 WAITING             rman channel=ORA_SBT_TAPE_2
```

2. Using operating system-level tools appropriate to your platform, kill the hung sessions. For example, on Solaris execute a `kill -9` command:

   ```
   % kill -9 8642 8374
   ```

   On Windows, there is a command-line utility called `ORAKILL` which lets you kill a specific thread in this situation. From a command prompt, run the following command:

   ```
   orakill sid thread_id
   ```

   where *`sid`* identifies the database instance to target, and the *`thread_id`* is the `SPID` value from the query in step 1.

3. Check that the media manager also clears its processes. If any remain, the next backup or restore operation may hang again, due to the previous hang. In some media managers, the only solution is to shut down and restart the media manager. If the documentation from the media manager does not provide the needed information, contact technical support for the media manager.

   See Also: Your operating system specific documentation for the relevant commands

RMAN Troubleshooting Scenarios

This section contains these topics:

- After Installation of Media Manager, RMAN Channel Allocation Fails: Scenario
- Backup Job Is Hanging: Scenario
- RMAN Fails to Start RPC Call: Scenario
- Backup Fails with Invalid RECID Error: Scenario

After Installation of Media Manager, RMAN Channel Allocation Fails: Scenario

In this scenario, you install and test the media manager as explained in "Configuring RMAN to Make Backups to a Media Manager" on page 6-5, but you still cannot make RMAN back up to tape. For example, after allocating the `sbt` channel, you receive an error stack similar to the following:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of allocate command on c1 channel at 08/29/2001 17:16:54
ORA-19554: error allocating device, device type: SBT_TAPE, device name:
ORA-27211: Failed to load Media Management Library
Additional information: 25
```

The most important line of the error output is the ORA-27211 error. It indicates the basic problem, that the media management library could not be loaded. Typically, there is no need to refer to the trace file or sbtio.log in such a case.

After Installation of Media Manager, RMAN Channel Allocation Fails: Diagnosis

The `ORA-27211` error indicates that the channel allocation is failing because the database is not loading the media management library. If the channel allocation fails, then the database generates a trace file in the `USER_DUMP_DEST` location that contains the error that caused the channel allocation to fail. The trace file should have the complete path name of the media management library loaded by the database as well as any other media manager errors or operating system errors. For example, the trace file on UNIX may be called something like

/oracle/rdbms/log/prod1_ora_16226.trc, and may contain information such as the following:

```
*** 2001-08-29 17:16:54.385
SKGFQ OSD: Error in function sbtinit on line 2396
SKGFQ OSD: Look for SBT Trace messages in file /oracle/rdbms/log/sbtio.log
SBT Initialize failed for oracle.static
```

The last line of this output indicates that Oracle is loading the default static library instead of the media management library that you installed.

You may find more detailed information in the file sbtio.log, as described in the error message. Note, however, that writing SBT trace messages is the responsibility of the media management software, not the Oracle database or RMAN. The media management vendor may not have implemented the writing of trace messages in a particular situation. Contact the media management vendor for details about the trace messages written to sbtio.log.

To test the loading of the media management library, try allocating a channel by using the PARMS parameter SBT_LIBRARY to force the loading of the media management library. For example, if your library is called /vendor/lib/some_mm_lib.so, then run a command such as the following, making sure to specify whatever PARMS settings are required by your media manager:

```
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt
    PARMS='SBT_LIBRARY=/vendor/lib/some_mm_lib.so',
    'ENV=(NSR_SERVER=tape_svr,NSR_CLIENT=oracleclnt,NSR_GROUP=oracle_tapes)';
}
```

If the channel allocation fails, then check the trace file again to see whether you can learn anything new. If the channel allocation with SBT_LIBRARY succeeds, but an ordinary sbt channel allocation fails, then the database is probably trying to load a library other than the one you installed. By default, the database expects to find the media management library at $ORACLE_HOME/lib/libobk.so on UNIX, or %ORACLE_HOME%/bin/orasbt.dll on NT. You may have more than one library in the operating system path, and the database is loading the wrong one.

After Installation of Media Manager, RMAN Channel Allocation Fails: Solution

If the problem is that the database is not loading the correct library, then make sure that the library is named correctly in the SBT_LIBRARY parameter.

See Also: *Oracle Database Recovery Manager Reference* for descriptions of the legal PARMS parameters

Backup Job Is Hanging: Scenario

In this scenario, an RMAN backup job starts as normal and then pauses inexplicably:

```
Recovery Manager: Release 10.1.0.2.0 - Production

Copyright (c) 1995, 2003, Oracle.  All rights reserved.

connected to target database: TRGT
connected to recovery catalog database

RMAN> BACKUP TABLESPACE SYSTEM, tools;

allocated channel: t1
channel t1: sid=16 devtype=SBT_TAPE

channel t1: starting datafile backupset
set_count=15 set_stamp=338309600
channel t1: including datafile 2 in backupset
channel t1: including datafile 1 in backupset
channel t1: including current controlfile in backupset
# Hanging here for 30 minutes now
```

Backup Job Is Hanging: Diagnosis

If a backup job is hanging, that is, not proceeding, then several scenarios are possible:

- A server-side or media management error occurred.
- RMAN is waiting for an event such as the insertion of a new cassette into the tape device.

Query `sbt` wait events to gain more information. For example, run the following query on the target instance:

```
COLUMN EVENT FORMAT a10
COLUMN SECONDS_IN_WAIT FORMAT 999
COLUMN STATE FORMAT a20
COLUMN CLIENT_INFO FORMAT a30
```

```
SELECT p.SPID, EVENT, SECONDS_IN_WAIT AS SEC_WAIT,
       STATE, CLIENT_INFO
FROM V$SESSION_WAIT sw, V$SESSION s, V$PROCESS p
WHERE sw.EVENT LIKE 'sbt%'
      AND s.SID=sw.SID
      AND s.PADDR=p.ADDR
;
```

Examine the SQL output to determine which sbt functions are waiting. For example, the output may be as follows:

```
SPID EVENT        SEC_WAIT STATE               CLIENT_INFO
---- ---------- ---------- ------------------- ------------------------------
8642 sbtbackup        1500 WAITING             rman channel=ORA_SBT_TAPE_1
```

Backup Job Is Hanging: Solution

Because the causes of a hung backup job can be varied, so are the solutions. For example, backup jobs often hang simply because the tape device has completely filled the current cassette and is waiting for a new tape to be inserted. Ideally, the query of the `sbt` wait events should indicate the problem.

In this example, a single sbtbackup has taken 1500 seconds, so RMAN is waiting on the media manager to finish its write operation. Check that the media manager is functioning normally, and contact the media management vendor's technical support for assistance.

If the `sbt` wait event query is unhelpful, then examine media manager process, log, and trace files for signs of abnormal termination or other errors (refer to the description of message files in "Identifying Types of Message Output" on page 15-2).

See Also: "Terminating an RMAN Session: Basic Steps" on page 15-15 to learn how to kill an RMAN session that is hanging

---

RMAN Fails to Start RPC Call: Scenario

In this scenario, you run a backup job and receive message output similar to the following:

```
channel c8: including datafile number 47 in backupset
RPC call appears to have failed to start on channel c9
RPC call ok on channel c9
channel c3: including datafile number 18 in backupset
```

RMAN Fails to Start RPC Call: Diagnosis

The RPC `call appears to have failed` message does not usually indicate a problem. The message indicates one of the following:

- The target database instance is slow.
- A timing problem occurred.

Timing problems occur in this way. When RMAN begins an RPC, it checks the `V$SESSION` performance view. The RPC updates the information in the view to indicate when it starts and finishes. Sometimes RMAN checks `V$SESSION` before the RPC has indicated it has started, which in turn generates the following message:

```
RPC call appears to have failed
```

If a message stating "RPC `call ok`" does not appear in the output immediately following the message stating "RPC `call appears to have failed`", then the backup job encountered an internal problem. Contact Oracle Support for further assistance.

Backup Fails with Invalid RECID Error: Scenario

In this scenario, you attempt a backup and receive the following error messages:

```
RMAN-3014: Implicit resync of recovery catalog failed
RMAN-6038: Recovery catalog package detected an error
RMAN-20035: Invalid high RECID error
```

Backup Fails with Invalid RECID Error: Diagnosis

In one common scenario, you restore a backup control file created through a non-Oracle mechanism, and then open the database without the `RESETLOGS` option. If you had created the backup control file through the RMAN `BACKUP` command or the SQL `ALTER DATABASE BACKUP CONTROLFILE` statement, then the database would have required you to reset the online logs.

The control file and the recovery catalog are now not synchronized. The database control file is older than the recovery catalog, because at one time the recovery catalog resynchronized with the old current control file, and now the database is using a backup control file. RMAN detects that the control file currently in use is older than the control file previously used to resynchronize.

Another common scenario occurs when you attempt to copy the target database to a new machine as follows:

1. On machine 1, you shut down the database and make a copy of the control file with an operating system utility. You do not use CATALOG to add this control file copy to the repository.
2. You transfer the control file copy to machine 2.
3. On machine 2, you create a new initialization parameter file and new database instance.
4. You mount the control file copy on machine 2. The database does not recognize the control file as a backup control file: to the database it looks like the current control file.
5. You start RMAN and connect to the new target database and the recovery catalog on machine 2. Because the control file was not created with RMAN and was not cataloged as a control file copy, RMAN sees the database on machine 2 as the database on machine 1.
6. You restore and recover database the new database on machine 2 and then open it. As a consequence, various records are added to the recovery catalog during the restore and recovery. For example, the highest RECID in the recovery catalog moves from 90 to 100.
7. On machine 1, you start RMAN and connect to the original target database and recovery catalog. The recovery catalog indicates that the highest RECID is 100, but the control file indicates that the highest RECID is 90. The control file RECID should always be greater than or equal to the recovery catalog RECID, so RMAN issues RMAN-20035.

Backup Fails with Invalid RECID Error: Solution 1

This solution is safest and is strongly recommended. It preserves the control file, so that the historical information about the database stored in the control file continues to be available after the procedure.

To reset the database with RMAN:

1. Connect to the target database with SQL*Plus. For example, enter:

```
% sqlplus '/ AS SYSDBA'
```

2. Mount the database if it is not already mounted. For example, enter:

```
ALTER DATABASE MOUNT;
```

3. Start cancel-based recovery by using the backup control file, then cancel it. The reason for canceling is that the USING BACKUP CONTROLFILE clause stamps the controlfile as a backup, which then permits OPEN RESETLOGS. For example, enter:

```
ALTER DATABASE RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE;
ALTER DATABASE RECOVER CANCEL;
```

4. Use RMAN to connect to the target database and recovery catalog. For example, enter:

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb
```

5. Open the database with the RESETLOGS option. For example, enter:

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

6. Take new backups so that you can recover the database if necessary. For example, enter:

```
BACKUP DATABASE PLUS ARCHIVELOG;
```

Backup Fails with Invalid RECID Error: Solution 2

This solution is similar to the previous one, but does require that you re-create your control file. It is better-suited for the case in which you are copying your database to a second system, where you may not want to keep the history from the control file for the copy of the database on the second system, or where you might drop a few datafiles or change the online logs by editing your control file.

To create the control file with SQL*Plus:

1. Connect to the target database with SQL*Plus. For example, enter:

```
% sqlplus 'SYS/oracle@trgt AS SYSDBA'
```

2. Mount the database if it is not already mounted:

```
SQL> ALTER DATABASE MOUNT;
```

3. Back up the control file to a trace file:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

4. Edit the trace file as necessary. The relevant section of the trace file looks something like the following:

```
# The following commands will create a new control file and use it
# to open the database.
# Data used by the recovery manager will be lost. Additional logs may
# be required for media recovery of offline data files. Use this
# only if the current version of all online logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "TRGT" NORESETLOGS  ARCHIVELOG
--  STANDBY DATABASE CLUSTER CONSISTENT AND UNPROTECTED
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 226
LOGFILE
  GROUP 1 '/oracle/oradata/trgt/redo01.log'  SIZE 25M,
  GROUP 2 '/oracle/oradata/trgt/redo02.log'  SIZE 25M,
  GROUP 3 '/oracle/oradata/trgt/redo03.log'  SIZE 500K
-- STANDBY LOGFILE
DATAFILE
  '/oracle/oradata/trgt/system01.dbf',
  '/oracle/oradata/trgt/undotbs01.dbf',
  '/oracle/oradata/trgt/cwmlite01.dbf',
  '/oracle/oradata/trgt/drsys01.dbf',
  '/oracle/oradata/trgt/example01.dbf',
  '/oracle/oradata/trgt/indx01.dbf',
  '/oracle/oradata/trgt/tools01.dbf',
  '/oracle/oradata/trgt/users01.dbf'
CHARACTER SET WE8DEC
;
# Take files offline to match current control file.
ALTER DATABASE DATAFILE '/oracle/oradata/trgt/tools01.dbf' OFFLINE;
ALTER DATABASE DATAFILE '/oracle/oradata/trgt/users01.dbf' OFFLINE;
# Configure RMAN configuration record 1
VARIABLE RECNO NUMBER;
EXECUTE :RECNO := SYS.DBMS_BACKUP_RESTORE.SETCONFIG('CHANNEL','DEVICE TYPE
DISK DEBUG 255');
# Recovery is required if any of the datafiles are restored backups,
# or if the last shutdown was not normal or immediate.
RECOVER DATABASE
# All logs need archiving and a log switch is needed.
ALTER SYSTEM ARCHIVE LOG ALL;
# Database can now be opened normally.
ALTER DATABASE OPEN;
# Commands to add tempfiles to temporary tablespaces.
# Online tempfiles have complete space information.
```

```
# Other tempfiles may require adjustment.
ALTER TABLESPACE TEMP ADD TEMPFILE '/oracle/oradata/trgt/temp01.dbf' REUSE;
# End of tempfile additions.
```

5. Shut down the database:

```
SHUTDOWN IMMEDIATE
```

6. Execute the script to create the control file, recover (if necessary), archive the logs, and open the database:

```
STARTUP NOMOUNT
CREATE CONTROLFILE ...;
EXECUTE ...;
RECOVER DATABASE
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER DATABASE OPEN ...;
```

7. If you intend to keep and continue using this copy of the database, use the DBNEWID utility to change the name and DBID of the new database as needed.

Caution: If you do *not* open with the RESETLOGS option, then two copies of an archived redo log for a given log sequence number may exist—even though these two copies have completely different contents. For example, one log may have been created on the original host and the other on the new host. If you accidentally confuse the logs during a media recovery, then the database will be corrupted but Oracle and RMAN cannot detect the problem.

Backup Fails Because of Control File Enqueue: Scenario

In this scenario, a backup job fails because RMAN cannot make a snapshot control file. The message stack is as follows:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of backup command at 08/30/2001 22:48:44
ORA-00230: operation disallowed: snapshot controlfile enqueue unavailable
```

Backup Fails Because of Control File Enqueue: Diagnosis

When RMAN needs to back up or resynchronize from the control file, it first creates a **snapshot** or consistent image of the control file. If one RMAN job is already

backing up the control file while another needs to create a new snapshot control file, then you may see the following message:

```
waiting for snapshot controlfile enqueue
```

Under normal circumstances, a job that must wait for the control file enqueue waits for a brief interval and then successfully obtains the enqueue. RMAN makes up to five attempts to get the enqueue and then fails the job. The conflict is usually caused when two jobs are both backing up the control file, and the job that first starts backing up the control file waits for service from the media manager.

To determine which job is holding the conflicting enqueue:

1. After you see the first message stating "`RMAN-08512: waiting for snapshot controlfile enqueue`", start a new SQL*Plus session on the target database:

   ```
   % sqlplus 'SYS/oracle@trgt AS SYSDBA'
   ```

2. Execute the following query to determine which job is causing the wait:

   ```
   SELECT s.SID, USERNAME AS "User", PROGRAM, MODULE,
          ACTION, LOGON_TIME "Logon", l.*
   FROM V$SESSION s, V$ENQUEUE_LOCK l
   WHERE l.SID = s.SID
   AND l.TYPE = 'CF'
   AND l.ID1 = 0
   AND l.ID2 = 2;
   ```

   You should see output similar to the following (the output in this example has been truncated):

   ```
   SID User Program              Module                    Action
   Logon
   --- ---- -------------------- ------------------ --------------- ---------
   9 SYS rman@h13 (TNS V1-V3) backup full datafile: c10000210 STARTED 21-JUN-01
   ```

Backup Fails Because of Control File Enqueue: Solution

Commonly, enqueue situations occur when a job is writing to a tape drive, but the tape drive is waiting for new tape to be inserted. If you start a new job in this situation, then you will probably receive the enqueue message because the first job cannot complete until the new tape is loaded.

After you have determined which job is creating the enqueue, you can do one of the following:

- Wait until the job holding the enqueue completes
- Cancel the current job and restart it after the job holding the enqueue completes
- Cancel the job creating the enqueue

RMAN Fails to Delete All Archived Logs: Scenario

In this scenario, the database archives automatically to two directories: *ORACLE_HOME*/oradata/trgt/arch and *ORACLE_HOME*/oradata/trgt/arch2. You tell RMAN to perform a backup and delete the input archived redo logs afterward in the following script:

```
BACKUP ARCHIVELOG ALL DELETE INPUT;
```

You then run a crosscheck to make sure the logs are gone and find the following:

```
CROSSCHECK ARCHIVELOG ALL;

validation succeeded for archived log
archivelog filename=/oracle/oradata/trgt/arch2/archive1_964.arc recid=19
stamp=368726072
```

RMAN deleted one set of logs but not the other.

RMAN Fails to Delete All Archived Logs: Diagnosis

This problem is not an error. When you specify DELETE INPUT without the ALL keyword, RMAN deletes only one copy of each input log. Even if you archive to five destinations, RMAN deletes logs from only one directory.

RMAN Fails to Delete All Archived Logs: Solution

To force RMAN to delete all existing archived redo logs, use the DELETE ALL INPUT clause of the BACKUP command. For example, enter:

```
BACKUP ARCHIVELOG ALL DELETE ALL INPUT;
```

Backup Fails Because RMAN Cannot Locate an Archived Log: Scenario

In this scenario, you schedule regular backups of the archived redo logs. The next time you make a backup, you receive this error:

```
RMAN-6089:  archive log NAME not found or out of sync with catalog
```

Backup Fails Because RMAN Cannot Locate an Archived Log: Diagnosis

This problem occurs when the archived log that RMAN is looking for cannot be accessed by RMAN, or the recovery catalog needs to be resynchronized. Often, this error occurs when you delete archived logs with an operating system command, which means that RMAN is unaware of the deletion. The `RMAN-6089` error occurs because RMAN attempts to back up a log that the repository indicates still exists.

Backup Fails Because RMAN Cannot Locate an Archived Log: Solution

Make sure that the archived logs exists in the specified directory and that the RMAN catalog is synchronized. Check the following:

1. Make sure the archived log file that is specified by the `RMAN-6089` error exists in the correct directory.
2. Check that the operating system permissions are correct for the archived log (`owner = oracle`, `group = DBA`) to make sure that RMAN can access the file.
3. If the file appears to be correct, then try synchronizing the catalog by running the following command from the RMAN prompt:

   ```
   RESYNC CATALOG;
   ```

If you know that the logs are unavailable because you deleted them by using an operating system utility, then run the following command at the RMAN prompt to update RMAN metadata:

```
CROSSCHECK ARCHIVELOG ALL;
```

It is always better to use RMAN to delete logs than to use an operating system utility. The easiest method to remove unwanted logs is to specify the `DELETE INPUT` option when backing up archived logs. For example, enter:

```
BACKUP DEVICE TYPE sbt
  ARCHIVELOG ALL
  DELETE ALL INPUT;
```

RMAN Does Not Recognize Character Set Name: Scenario

In this scenario, you are connected to the target database while it is not open and attempting to perform an RMAN operation. You receive the following error:

```
PLS-00553: character set name is not recognized
```

RMAN Does Not Recognize Character Set Name: Diagnosis

Typically, this message means that the character set in the client environment, that is, the environment in which you are running the RMAN client, is different from the character set in the target database environment.

RMAN Does Not Recognize Character Set Name: Solution

1. Query the target database to determine the value of the NLS_CHARACTERSET parameter. For example, run this query:

   ```
   SQL> SELECT VALUE FROM V$NLS_PARAMETERS WHERE PARAMETER='NLS_CHARACTERSET';
   ```

2. Set the character set environment variable in the client to the same value as the variable in the server. For example, you can set the NLS_LANG environment variable on a UNIX system as follows:

   ```
   % setenv NLS_LANG american_america.we8dec
   % setenv NLS_DATE_FORMAT "MON DD YYYY HH24:MI:SS"
   ```

If the connection is made througfh a listener, then the listener must be started with the correct Globalization Support settings. Otherwise, the spawned connections inherit the incorrect Globalization Support settings from the listener.

RMAN Denies Logon to Target Database: Scenario

RMAN fails with ORA-01031 (insufficient privileges) or ORA-01017 (invalid username/password) errors when trying to connect to the target database:

```
% rman
Recovery Manager: Release 10.1.0.2.0 - Production

Copyright (c) 1995, 2003, Oracle.  All rights reserved.

RMAN> CONNECT TARGET sys/mypass@inst1

RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
ORA-01031: insufficient privileges
```

RMAN Denies Logon to Target Database: Diagnosis

RMAN automatically requests a connection to the target database as SYSDBA. In order to connect to the target as SYSDBA, you must do one of the following:

- Be part of the operating system DBA group with respect to the target database (that is, have the ability to connect with SYSDBA privileges to the target database without a password).
- Create a password file with the orapwd command and the initialization parameter REMOTE_LOGIN_PASSWORDFILE.
- Make sure you are connecting with the correct username and password.

If the target database does not have a password file, then the user you are logged in as must be validated with operating system authentication.

RMAN Denies Logon to Target Database: Solution

Either create a password file for the target database or add yourself to the administrator list in the operating system.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to create a password file

Database Duplication Fails Because of Missing Log: Scenario

In this scenario, you attempt to duplicate a database with the DUPLICATE command, but receive the following error stack:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of Duplicate Db command at 09/04/2001 12:11:29
RMAN-03015: error occurred in stored script Memory Script
RMAN-06053: unable to perform media recovery because of missing log
RMAN-06025: no backup of log thread 1 seq 16 scn 145858 found to restore
```

Database Duplication Fails Because of Missing Log: Diagnosis

The problem is that RMAN is not able to apply all the archived logs needed for complete recovery. For example, if you only backed up logs through sequence 15, but the most recent archived log is sequence 16, then DUPLICATE fails.

Database Duplication Fails Because of Missing Log: Solution

When creating the duplication script, use the SET UNTIL command to specify a log sequence number for incomplete recovery. For example, to terminate recovery after applying log sequence 15, enter:

```
RUN
```

```
{
  SET UNTIL SEQUENCE 16 THREAD 1;  # recovers up to but not including log 16
  DUPLICATE TARGET DATABASE TO 'dupdb';
}
```

See Also: "Creating Duplicate of the Database at a Past Point in Time: Example" on page 11-23 for more information about performing incomplete recovery during the duplication operation

Duplication Fails with Multiple RMAN-06023 Errors: Scenario

In this scenario, you back up the database, then run the DUPLICATE command. You receive the following error stack:

```
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03002: failure of Duplicate Db command at 09/04/2001 13:55:11
RMAN-03015: error occurred in stored script Memory Script
RMAN-06026: some targets not found - aborting restore
RMAN-06023: no backup or copy of datafile 8 found to restore
RMAN-06023: no backup or copy of datafile 7 found to restore
RMAN-06023: no backup or copy of datafile 6 found to restore
RMAN-06023: no backup or copy of datafile 5 found to restore
RMAN-06023: no backup or copy of datafile 4 found to restore
RMAN-06023: no backup or copy of datafile 3 found to restore
RMAN-06023: no backup or copy of datafile 2 found to restore
RMAN-06023: no backup or copy of datafile 1 found to restore
```

Duplication Fails with Multiple RMAN-06023 Errors: Diagnosis

The DUPLICATE command recovers to archived redo logs, but cannot recover into online redo logs. Thus, if the restored backup cannot be made consistent without applying the online redo logs, then duplication fails with RMAN-06023 errors because RMAN is looking for backups created *before* the most recent archived log.

Duplication Fails with Multiple RMAN-06023 Errors: Solution

After backing up the source database, archive and back up the current redo log:

```
RMAN> SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
RMAN> BACKUP ARCHIVELOG ALL;
```

This archives all records in the online redo logs so that RMAN can now recover the backup by applying the most recent archived redo log.

UNKNOWN Database Name Appears in Recovery Catalog: Scenario

In this scenario, you list the database incarnations registered in the recovery catalog and see a database with the name UNKNOWN:

```
LIST INCARNATION OF DATABASE;

RMAN-03022: compiling command: list
List of Database Incarnations
DB Key  Inc Key   DB Name   DB ID       STATUS    Reset SCN    Reset Time
------- -------   -------   ------      ------    ----------   ----------
56      57        TRGT      4052472287  CURRENT   1            Sep 03 2001 06:45:51
1       19        UNKNOWN   4141147584  PARENT    1            Jan 08 2001 14:47:28
.
.
.
```

UNKNOWN Database Name Appears in Recovery Catalog: Diagnosis

One way you get the DB_NAME of UNKNOWN is when you register a database that was once opened with the RESETLOGS option. The DB_NAME can be changed during a RESETLOGS operation, so RMAN does not know what the DB_NAME was for those old incarnations of the database because it was not registered in the recovery catalog at the time. Consequently, RMAN sets the DB_NAME column to UNKNOWN when creating the DBINC record.

UNKNOWN Database Name Appears in Recovery Catalog: Solution

The UNKNOWN name entry is expected behavior after a RESETLOGS operation. You should *not* attempt to remove UNKNOWN entries from the recovery catalog.

Part III

Performing User-Managed Backup and Recovery

The following chapters describe how to perform backup and recovery without using Recovery Manager. This part of the book contains these chapters:

16

Making User-Managed Backups

If you do not use Recovery Manager (RMAN), then you can make backups of your database files with user-managed methods.

This chapter contains the following sections:

Querying V$ Views to Obtain Backup Information

Before making a backup, you must identify all the files in your database and decide what to back up. Several V$ views can provide the necessary information.

Listing Database Files Before a Backup

Use V$DATAFILE, V$LOGFILE and V$CONTROLFILE to identify the datafiles, log files and control files for your database. This same procedure works whether you named these files manually or allowed Oracle Managed Files to name them.

To list datafiles, online redo logs, and control files:

1. Start SQL*Plus and query V$DATAFILE to obtain a list of datafiles. For example, enter:

   ```
   SQL> SELECT NAME FROM V$DATAFILE;
   ```

   You can also join the V$TABLESPACE and V$DATAFILE views to obtain a listing of datafiles along with their associated tablespaces:

   ```
   SELECT t.NAME "Tablespace", f.NAME "Datafile"
     FROM V$TABLESPACE t, V$DATAFILE f
     WHERE t.TS# = f.TS#
     ORDER BY t.NAME;
   ```

2. Obtain the filenames of online redo log files by querying the V$LOGFILE view. For example, issue the following query:

   ```
   SQL> SELECT MEMBER FROM V$LOGFILE;
   ```

3. Obtain the filenames of the current control files by querying the V$CONTROLFILE view. For example, issue the following query:

   ```
   SQL> SELECT NAME FROM V$CONTROLFILE;
   ```

   Note that you only need to back up one copy of a multiplexed control file.

4. If you plan to take a control file backup with the ALTER DATABASE BACKUP CONTROLFILE TO '*filename*' statement, then save a list of all datafiles and online redo log files with the control file backup. Because the current database structure may not match the database structure at the time a given control file backup was created, saving a list of files recorded in the backup control file can aid the recovery procedure.

Determining Datafile Status for Online Tablespace Backups

To check whether a datafile is part of a current online tablespace backup, query the V$BACKUP view.

This view is useful only for user-managed online tablespace backups, because neither RMAN backups nor offline tablespace backups require the datafiles of a tablespace to be in backup mode.

The V$BACKUP view is most useful when the database is open. It is also useful immediately after an instance failure because it shows the backup status of the files at the time of the failure. Use this information to determine whether you have left any tablespaces in backup mode.

V$BACKUP is not useful if the control file currently in use is a restored backup or a new control file created after the media failure occurred. A restored or re-created control file does not contain the information the database needs to populate V$BACKUP accurately. Also, if you have restored a backup of a file, this file's STATUS in V$BACKUP reflects the backup status of the older version of the file, not the most current version. Thus, this view can contain misleading data about restored files.

For example, the following query displays which datafiles are currently included in a tablespace that has been placed in backup mode:

```
SELECT t.name AS "TB_NAME", d.file# as "DF#", d.name AS "DF_NAME", b.status
FROM V$DATAFILE d, V$TABLESPACE t, V$BACKUP b
WHERE d.TS#=t.TS#
AND b.FILE#=d.FILE#
AND b.STATUS='ACTIVE'
/
```

The following sample output shows that the `tools` and `users` tablespaces currently have ACTIVE status:

```
TB_NAME                DF#        DF_NAME                          STATUS
---------------------  ---------- -------------------------------  ------
TOOLS                  7          /oracle/oradata/trgt/tools01.dbf  ACTIVE
USERS                  8          /oracle/oradata/trgt/users01.dbf  ACTIVE
```

In the STATUS column, NOT ACTIVE indicates that the file is not currently in backup mode (that is, you have not executed the ALTER TABLESPACE . . . BEGIN BACKUP or ALTER DATABASE BEGIN BACKUP statement), whereas ACTIVE indicates that the file is currently in backup mode.

Making User-Managed Backups of the Whole Database

You can make a whole database backup of all files in a database after the database has been shut down with the NORMAL, IMMEDIATE, or TRANSACTIONAL options. A whole database backup taken while the database is open or after an instance failure or SHUTDOWN ABORT is inconsistent. In such cases, the files are inconsistent with respect to the checkpoint SCN.

You can make a whole database backup if a database is operating in either ARCHIVELOG or NOARCHIVELOG mode. If you run the database in NOARCHIVELOG mode, however, the backup must be consistent; that is, you must shut down the database cleanly before the backup.

The set of backup files that results from a consistent whole database backup is consistent because all files are checkpointed to the same SCN. You can restore the consistent database backup without further recovery. After restoring the backup files, you can perform additional recovery steps to recover the database to a more current time if the database is operated in ARCHIVELOG mode. Also, you can take inconsistent whole database backups if your database is in ARCHIVELOG mode.

Control files play a crucial role in database restore and recovery. For databases running in ARCHIVELOG mode, Oracle Corporation recommends that you back up control files with the ALTER DATABASE BACKUP CONTROLFILE TO '*filename*' statement.

> **See Also:** "Making User-Managed Backups of the Control File" on page 16-14 for more information about backing up control files

Making Consistent Whole Database Backups

This section describes how to back up the database with an operating system utility.

To make a consistent whole database backup:

1. If the database is open, use SQL*Plus to shut down the database with the NORMAL, IMMEDIATE, or TRANSACTIONAL options.
2. Use an operating system utility to make backups of all datafiles as well as all control files specified by the CONTROL_FILES parameter of the initialization parameter file. Also, back up the initialization parameter file and other Oracle product initialization files. To find these files, do a search for *.ora starting in your Oracle home directory and recursively search all of its subdirectories.

   For example, you can back up the datafiles, control files and archived logs to /disk2/backup as follows:

```
% cp $ORACLE_HOME/oradata/trgt/*.dbf /disk2/backup
% cp $ORACLE_HOME/oradata/trgt/arch/* /disk2/backup/arch
```

3. Restart the database. For example, enter:

```
SQL> STARTUP
```

> **See Also:** *Oracle Database Administrator's Guide* for more information on starting up and shutting down a database

Making User-Managed Backups of Offline Tablespaces and Datafiles

You can back up all or some of the datafiles of an individual tablespace while the tablespace is offline. All other tablespaces of the database can remain open and available for systemwide use. You must have the DBA privilege or have the MANAGE TABLESPACE system privilege to take tablespaces offline and online.

Note the following guidelines when backing up offline tablespaces:

- You cannot offline the SYSTEM tablespace or a tablespace with active rollback segments. The following procedure cannot be used for such tablespaces.
- Assume that a table is in tablespace `Primary` and its index is in tablespace `Index`. Taking tablespace `Index` offline while leaving tablespace `Primary` online can cause errors when DML is issued against the indexed tables located in `Primary`. The problem only manifests when the access method chosen by the optimizer needs to access the indexes in the `Index` tablespace.

To back up offline tablespaces:

1. Before beginning a backup of a tablespace, identify the tablespace's datafiles by querying the DBA_DATA_FILES view. For example, assume that you want to back up the `users` tablespace. Enter the following in SQL*Plus:

```
SELECT TABLESPACE_NAME, FILE_NAME
  FROM SYS.DBA_DATA_FILES
  WHERE TABLESPACE_NAME = 'USERS';

TABLESPACE_NAME                FILE_NAME
------------------------------ --------------------------------
USERS                          /oracle/oradata/trgt/users01.dbf
```

   In this example, `/oracle/oradata/trgt/users01.dbf` is a fully specified filename corresponding to the datafile in the `users` tablespace.

2. Take the tablespace offline using normal priority if possible because it guarantees that you can subsequently bring the tablespace online without having to recover it. For example:

```
SQL> ALTER TABLESPACE users OFFLINE NORMAL;
```

3. Back up the offline datafiles. For example:

```
% cp /oracle/oradata/trgt/users01.dbf /d2/users01_‘date "+%m_%d_%y"‘.dbf
```

4. Bring the tablespace online. For example:

```
ALTER TABLESPACE users ONLINE;
```

> **Note:** If you took the tablespace offline using temporary or immediate priority, then you cannot bring the tablespace online unless you perform tablespace recovery.

5. Archive the unarchived redo logs so that the redo required to recover the tablespace backup is archived. For example, enter:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Making User-Managed Backups of Online Tablespaces and Datafiles

You can back up all or only specific datafiles of an online tablespace while the database is open. The procedure differs depending on whether the online tablespace is read / write or read-only.

> **Note:** You should not back up temporary tablespaces.

Making User-Managed Backups of Online Read/Write Tablespaces

You must put a read / write tablespace in backup mode to make user-managed datafile backups when the tablespace is online and the database is open. The ALTER TABLESPACE ... BEGIN BACKUP statement places a tablespace in backup mode. In backup mode, the database copies whole changed data blocks into the redo stream. After you take the tablespace out of backup mode with the ALTER TABLESPACE . . . END BACKUP or ALTER DATABASE END BACKUP statement, the database advances the datafile header to the current database checkpoint.

When restoring a datafile backed up in this way, the database asks for the appropriate set of redo log files to apply if recovery be needed. The redo logs contain all changes required to recover the datafiles and make them consistent.

To back up online read/write tablespaces in an open database:

1. Before beginning a backup of a tablespace, identify all of the datafiles in the tablespace with the DBA_DATA_FILES data dictionary view. For example, assume that you want to back up the users tablespace. Enter the following:

```
SELECT TABLESPACE_NAME, FILE_NAME
FROM SYS.DBA_DATA_FILES
WHERE TABLESPACE_NAME = 'USERS';

TABLESPACE_NAME                FILE_NAME
------------------------------ --------------------
USERS                          /oracle/oradata/trgt/users01.dbf
USERS                          /oracle/oradata/trgt/users02.dbf
```

2. Mark the beginning of the online tablespace backup. For example, the following statement marks the start of an online backup for the tablespace users:

```
SQL> ALTER TABLESPACE users BEGIN BACKUP;
```

> **Caution:** If you do not use BEGIN BACKUP to mark the beginning of an online tablespace backup and wait for that statement to complete before starting your copies of online tablespaces, or then the datafile copies produced are not usable for subsequent recovery operations. Attempting to recover such a backup is risky and can return errors that result in inconsistent data. For example, the attempted recovery operation can issue a "fuzzy files" warning, and can lead to an inconsistent database that you cannot open.

3. Back up the online datafiles of the online tablespace with operating system commands. For example, UNIX users might enter:

```
% cp /oracle/oradata/trgt/users01.dbf /d2/users01_‘date "+%m_%d_%y"‘.dbf
% cp /oracle/oradata/trgt/users02.dbf /d2/users02_‘date "+%m_%d_%y"‘.dbf
```

4. After backing up the datafiles of the online tablespace, run the SQL statement ALTER TABLESPACE with the END BACKUP option. For example, the following statement ends the online backup of the tablespace users:

```
SQL> ALTER TABLESPACE users END BACKUP;
```

5. Archive the unarchived redo logs so that the redo required to recover the tablespace backup is archived. For example, enter:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

> **Caution:** If you fail to take the tablespace out of backup mode, then Oracle continues to write copies of data blocks in this tablespace to the online logs, causing performance problems. Also, you will receive an `ORA-01149` error if you try to shut down the database with the tablespaces still in backup mode.

Making Multiple User-Managed Backups of Online Read/Write Tablespaces

When backing up several online tablespaces, you can back them up either serially or in parallel. Use either of the following procedures depending on your needs.

Backing Up Online Tablespaces in Parallel

You can simultaneously create datafile copies of multiple tablespaces requiring backups in backup mode. Note, however, that by putting all tablespaces in online mode at once, you can generate large redo logs if there is heavy update activitiy on the affected tablespaces, because the redo must contain a copy of each changed data block in each changed datafile. Be sure to consider the size of the likely redo before using the procedure outlined here.

To back up online tablespaces in parallel:

1. Prepare all online tablespaces for backup by issuing all necessary `ALTER TABLESPACE` statements at once. For example, put tablespaces `users`, `tools`, and `indx` in backup mode as follows:

```
SQL> ALTER TABLESPACE users BEGIN BACKUP;
SQL> ALTER TABLESPACE tools BEGIN BACKUP;
SQL> ALTER TABLESPACE indx BEGIN BACKUP;
```

   If you are backing up all tablespaces, you might want to use this command:

```
SQL> ALTER DATABASE BEGIN BACKUP;
```

2. Back up all files of the online tablespaces. For example, a UNIX user might back up datafiles with the `*.dbf` suffix as follows:

```
% cp $ORACLE_HOME/oradata/trgt/*.dbf /disk2/backup/
```

3. Take the tablespaces out of backup mode as in the following example:

```
SQL> ALTER TABLESPACE users END BACKUP;
SQL> ALTER TABLESPACE tools END BACKUP;
SQL> ALTER TABLESPACE indx END BACKUP;
```

   Again, it you are handling all datafiles at once you can use the ALTER DATABASE command instead of ALTER TABLESPACE:

```
SQL> ALTER DATABASE END BACKUP;
```

4. Archive the online redo logs so that the redo required to recover the tablespace backups will be available for later media recovery. For example, enter:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Backing Up Online Tablespaces Serially

You can place all tablespaces requiring online backups in backup mode one at a time. Oracle Corporation recommends the serial backup option because it minimizes the time between `ALTER TABLESPACE ... BEGIN/END BACKUP` statements. During online backups, more redo information is generated for the tablespace because whole data blocks are copied into the redo log.

To back up online tablespaces serially:

1. Prepare a tablespace for online backup. For example, to put tablespace `users` in backup mode enter the following:

```
SQL> ALTER TABLESPACE users BEGIN BACKUP;
```

   In this case you probably do not want to use `ALTER DATABASE BEGIN BACKUP` to put all tablespaces in backup mode simultaneously, because of the unnecessary volume of redo log information generated for tablespaces in online mode.

2. Back up the datafiles in the tablespace. For example, enter:

```
% cp /oracle/oradata/trgt/users01.dbf /d2/users01_‘date "+%m_%d_%y"‘.dbf
```

3. Take the tablespace out of backup mode. For example, enter:

```
SQL> ALTER TABLESPACE users END BACKUP;
```

4. Repeat this procedure for each remaining tablespace.

5. Archive the unarchived redo logs so that the redo required to recover the tablespace backups is archived. For example, enter:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Ending a Backup After an Instance Failure or SHUTDOWN ABORT

The following situations can cause a tablespace backup to fail and be incomplete:

- The backup completed, but you did not run the ALTER TABLESPACE . . . END BACKUP statement.
- An instance failure or SHUTDOWN ABORT interrupted the backup.

Whenever crash recovery is required, if a datafile is in backup mode when an attempt is made to open it, then the database will not open the database until either a recovery command is issued, or the datafile is taken out of backup mode.

For example, the database may display a message such as the following at startup:

```
ORA-01113: file 12 needs media recovery
ORA-01110: data file 12: '/oracle/dbs/tbs_41.f'
```

If the database indicates that the datafiles for multiple tablespaces require media recovery because you forgot to end the online backups for these tablespaces, then so long as the database is mounted, running the ALTER DATABASE END BACKUP statement takes all the datafiles out of backup mode simultaneously.

In high availability situations, and in situations when no DBA is monitoring the database, the requirement for user intervention is intolerable. Hence, you can write a crash recovery script that does the following:

1. Mounts the database
2. Runs the ALTER DATABASE END BACKUP statement
3. Runs ALTER DATABASE OPEN, allowing the system to come up automatically

An automated crash recovery script containing ALTER DATABASE END BACKUP is especially useful in the following situations:

- All nodes in an Oracle Real Application Clusters (RAC) configuration fail.
- One node fails in a **cold failover cluster** (that is, a cluster that is *not* a RAC configuration in which the secondary node must mount and recover the database when the first node fails).

Alternatively, you can take the following manual measures after the system fails with tablespaces in backup mode:

- Recover the database and avoid issuing END BACKUP statements altogether.
- Mount the database, then run ALTER TABLESPACE . . . END BACKUP for each tablespace still in backup mode.

Ending Backup Mode with the ALTER DATABASE END BACKUP Statement

You can run the ALTER DATABASE END BACKUP statement when you have multiple tablespaces still in backup mode. The primary purpose of this command is to allow a crash recovery script to restart a failed system without DBA intervention. You can also perform the following procedure manually.

To take tablespaces out of backup mode simultaneously:

1. Mount but do not open the database. For example, enter:

```
SQL> STARTUP MOUNT
```

2. If performing this procedure manually (that is, not as part of a crash recovery script), query the V$BACKUP view to list the datafiles of the tablespaces that were being backed up before the database was restarted:

```
SQL>  SELECT * FROM V$BACKUP WHERE STATUS = 'ACTIVE';
FILE#      STATUS             CHANGE#    TIME
---------- ------------------ ---------- ---------
        12 ACTIVE                  20863 25-NOV-02
        13 ACTIVE                  20863 25-NOV-02
        20 ACTIVE                  20863 25-NOV-02
 3 rows selected.
```

3. Issue the ALTER DATABASE END BACKUP statement to take all datafiles currently in backup mode out of backup mode. For example, enter:

```
SQL> ALTER DATABASE END BACKUP;
```

   You can use this statement only when the database is mounted but not open. If the database is open, use ALTER TABLESPACE . . . END BACKUP or ALTER DATABASE DATAFILE . . . END BACKUP for each affected tablespace or datafile.

> **Caution:** Do not use ALTER DATABASE END BACKUP if you have restored any of the affected files from a backup.

Ending Backup Mode with the SQL*Plus RECOVER Command

The ALTER DATABASE END BACKUP statement is not the only way to respond to a failed online backup: you can also run the SQL*Plus RECOVER command. This method is useful when you are not sure whether someone has restored a backup, because if someone has indeed restored a backup, then the RECOVER command brings the backup up to date. Only run the ALTER DATABASE END BACKUP or ALTER TABLESPACE . . . END BACKUP statement if you are sure that the files are current.

Note: The RECOVER command method is slow because the database must scan redo generated from the beginning of the online backup.

To take tablespaces out of backup mode with the RECOVER command:

1. Mount the database. For example, enter:

```
SQL> STARTUP MOUNT
```

2. Recover the database as normal. For example, enter:

```
SQL> RECOVER DATABASE
```

3. Use the V$BACKUP view to confirm that there are no active datafiles:

```
SQL>  SELECT * FROM V$BACKUP WHERE STATUS = 'ACTIVE';
FILE#      STATUS             CHANGE#    TIME
---------- ------------------ ---------- ---------
0 rows selected.
```

See Also: Chapter 17, "Performing User-Managed Database Flashback and Recovery" for information on recovering a database

Making User-Managed Backups of Read-Only Tablespaces

When backing up an online read-only tablespace, you can simply back up the online datafiles. You do not have to place the tablespace in backup mode because the system is permitting changes to the datafiles.

If the set of read-only tablespaces is self-contained, then in addition to backing up the tablespaces with operating system commands, you can also export the tablespace metadata with the transportable tablespace functionality. In the event of

a media error or a user error (such as accidentally dropping a table in the read-only tablespace), you can transport the tablespace back into the database.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to transport tablespaces

To back up online read-only tablespaces in an open database:

1. Query the DBA_TABLESPACES view to determine which tablespaces are read-only. For example, run this query:

```
SELECT TABLESPACE_NAME, STATUS
FROM DBA_TABLESPACES
WHERE STATUS = 'READ ONLY';
```

2. Before beginning a backup of a read-only tablespace, identify all of the tablespace's datafiles by querying the DBA_DATA_FILES data dictionary view. For example, assume that you want to back up the `history` tablespace:

```
SELECT TABLESPACE_NAME, FILE_NAME
FROM SYS.DBA_DATA_FILES
WHERE TABLESPACE_NAME = 'HISTORY';

TABLESPACE_NAME                FILE_NAME
------------------------------ --------------------
HISTORY                        /oracle/oradata/trgt/history01.dbf
HISTORY                        /oracle/oradata/trgt/history02.dbf
```

3. Back up the online datafiles of the read-only tablespace with operating system commands. You do not have to take the tablespace offline or put the tablespace in backup mode because users are automatically prevented from making changes to the read-only tablespace. For example:

```
% cp $ORACLE_HOME/oradata/trgt/history*.dbf  /disk2/backup/
```

> **Note:** When restoring a backup of a read-only tablespace, take the tablespace offline, restore the datafiles, then bring the tablespace online. A backup of a read-only tablespace is still usable if the read-only tablespace is made read / write after the backup, but the restored backup will require recovery.

4. Optionally, export the metadata in the read-only tablespace. By using the transportable tablespace feature, you can quickly restore the datafiles and

import the metadata in case of media failure or user error. For example, export the metadata for tablespace `history` as follows:

```
% exp TRANSPORT_TABLESPACE=y TABLESPACES=(history) FILE=/disk2/backup/hs.dmp
```

> **See Also:** *Oracle Database Reference* for more information about the DBA_DATA_FILES and DBA_TABLESPACES views

Making User-Managed Backups of the Control File

Back up the control file of a database after making a structural modification to a database operating in ARCHIVELOG mode. To back up a database's control file, you must have the ALTER DATABASE system privilege.

Backing Up the Control File to a Binary File

The primary method for backing up the control file is to use a SQL statement to generate a binary file. A binary backup is preferable to a trace file backup because it contains additional information such as the archived log history, offline range for read-only and offline tablespaces, and backup sets and copies (if you use RMAN). Note that binary control file backups do not include tempfile entries.

To back up the control file after a structural change:

1. Make the desired change to the database. For example, you may create a new tablespace:

   ```
   CREATE TABLESPACE tbs_1 DATAFILE 'file_1.f' SIZE 10M;
   ```

2. Back up the database's control file, specifying a filename for the output binary file. The following example backs up a control file to `/disk1/backup/cf.bak`:

   ```
   ALTER DATABASE BACKUP CONTROLFILE TO '/disk1/backup/cf.bak' REUSE;
   ```

   Specify the REUSE option to make the new control file overwrite one that currently exists.

Backing Up the Control File to a Trace File

The TRACE option of the ALTER DATABASE BACKUP CONTROLFILE statement helps you manage and recover the control file. The TRACE option prompts the database to write SQL statements to the database's trace file rather than generate a binary

backup. The statements in the trace file start the database, re-create the control file, and recover and open the database appropriately.

To back up the control file to a trace file, mount or open the database and issue the following SQL statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

If you specify neither the RESETLOGS nor NORESETLOGS option in the SQL statement, then the resulting trace file contains versions of the control file for both RESETLOGS and NORESETLOGS options. Tempfile entries are included in the output using "ALTER TABLESPACE... ADD TEMPFILE" statements.

> **See Also:** "Recovery of Read-Only Files with a Re-Created Control File" on page 18-7 for special issues relating to read-only, offline normal, and temporary files included in CREATE CONTROLFILE statements

Backing Up the Control File to a Trace File: Example

Assume that you want to generate a script that re-creates the control file for the sales database. The database has these characteristics:

- Three threads are enabled, of which thread 2 is public and thread 3 is private.
- The redo logs are multiplexed into three groups of two members each.
- The database has the following datafiles:
  - /diska/prod/sales/db/filea.dbf (offline datafile in online tablespace)
  - /diska/prod/sales/db/database1.dbf (online in SYSTEM tablespace)
  - /diska/prod/sales/db/fileb.dbf (only file in read-only tablespace)

You issue the following statement to create a trace file containing a CREATE CONTROLFILE ... NORESETLOGS statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS;
```

You then edit the trace file to create a script that creates a new control file for the sales database based on the control file that was current when you generated the trace file. To avoid recovering offline normal or read-only tablespaces, edit them out of the CREATE CONTROLFILE statement in the trace file. When you open the database with the re-created control file, the dictionary check code will mark these

omitted files as MISSING. You can run an ALTER DATABASE RENAME FILE statement renames them back to their original filenames.

For example, you can edit the CREATE CONTROLFILE . . . NORESETLOGS script in the trace file as follows, renaming files labeled MISSING:

```
# The following statements will create a new control file and use it to open the
# database. Log history and RMAN metadata will be lost. Additional logs may be
# required for media recovery of offline datafiles. Use this only if the current
# version of all online logs are available.

STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE SALES NORESETLOGS ARCHIVELOG
     MAXLOGFILES 32
     MAXLOGMEMBERS 2
     MAXDATAFILES 32
     MAXINSTANCES 16
     MAXLOGHISTORY 1600
LOGFILE
     GROUP 1
       '/diska/prod/sales/db/log1t1.dbf',
       '/diskb/prod/sales/db/log1t2.dbf'
     )  SIZE 100K
    GROUP 2
       '/diska/prod/sales/db/log2t1.dbf',
        '/diskb/prod/sales/db/log2t2.dbf'
    ) SIZE 100K,
    GROUP 3
       '/diska/prod/sales/db/log3t1.dbf',
       '/diskb/prod/sales/db/log3t2.dbf'
    ) SIZE 100K
DATAFILE
    '/diska/prod/sales/db/database1.dbf',
    '/diskb/prod/sales/db/filea.dbf'
;

# This datafile is offline, but its tablespace is online. Take the datafile
# offline manually.
ALTER DATABASE DATAFILE '/diska/prod/sales/db/filea.dbf' OFFLINE;

# Recovery is required if any datafiles are restored backups,
# or if the most recent shutdown was not normal or immediate.
RECOVER DATABASE;

# All redo logs need archiving and a log switch is needed.
ALTER SYSTEM ARCHIVE LOG ALL;
```

```
# The database can now be opened normally.
ALTER DATABASE OPEN;

# The backup control file does not list read-only and normal offline tablespaces
# so that Oracle can avoid performing recovery on them. Oracle checks the data
# dictionary and finds information on these absent files and marks them
# 'MISSINGxxxx'. It then renames the missing files to acknowledge them without
# having to recover them.
ALTER DATABASE RENAME FILE 'MISSING0002'
     TO '/diska/prod/sales/db/fileb.dbf';
```

Making User-Managed Backups of Archived Redo Logs

To save disk space in your primary archiving location, you may want to back up archived logs to tape or to an alternative disk location. If you archive to multiple locations, then only back up one copy of each log sequence number.

To back up archived redo logs:

1. To determine which archived redo log files that the database has generated, query V$ARCHIVED_LOG. For example, run the following query:

   ```
   SELECT THREAD#,SEQUENCE#,NAME
   FROM V$ARCHIVED_LOG;
   ```

2. Back up one copy of each log sequence number by using an operating system utility. This example backs up all logs in the primary archiving location to a disk devoted to log backups:

   ```
   % cp $ORACLE_HOME/oracle/trgt/arch/* /disk2/backup/arch
   ```

   See Also: *Oracle Database Reference* for more information about the data dictionary views

Making User-Managed Backups in SUSPEND Mode

This section contains the following topics:

- About the Suspend / Resume Feature
- Making Backups in a Suspended Database

About the Suspend/Resume Feature

Some third-party tools allow you to mirror a set of disks or logical devices, that is, maintain an exact duplicate of the primary data in another location, and then **split the mirror**. Splitting the mirror involves separating the copies so that you can use them independently.

With the SUSPEND/RESUME functionality, you can suspend I/O to the database, then split the mirror and make a backup of the split mirror. By using this feature, which complements the backup mode functionality, you can suspend database I/Os so that no new I/O can be performed. You can then access the suspended database to make backups without I/O interference.

You do not need to use SUSPEND/RESUME to make split mirror backups in most cases, although it is necessary if your system requires the database cache to be free of **dirty buffers** before a volume can be split. Some RAID devices benefit from suspending writes while the split operation is occurring; your RAID vendor can advise you on whether your system would benefit from this feature.

The ALTER SYSTEM SUSPEND statement suspends the database by halting I/Os to datafile headers, datafiles, and control files. When the database is suspended, all pre-existing I/O operations can complete; however, any new database I/O access attempts are queued.

The ALTER SYSTEM SUSPEND and ALTER SYSTEM RESUME statements operate on the database and not just the instance. If the ALTER SYSTEM SUSPEND statement is entered on one system in a RAC configuration, then the internal locking mechanisms propagate the halt request across instances, thereby suspending I/O operations for all active instances in a given cluster.

Making Backups in a Suspended Database

After a successful database suspension, you can back up the database to disk or break the mirrors. Because suspending a database does not guarantee immediate termination of I/O, Oracle Corporation recommends that you precede the ALTER SYSTEM SUSPEND statement with a BEGIN BACKUP statement so that the tablespaces are placed in backup mode.

You must use conventional user-managed backup methods to back up split mirrors. RMAN cannot make database backups or copies because these operations require reading the datafile headers. After the database backup is finished or the mirrors are re-silvered, then you can resume normal database operations using the ALTER SYSTEM RESUME statement.

Backing up a suspended database without splitting mirrors can cause an extended database outage because the database is inaccessible during this time. If backups are taken by splitting mirrors, however, then the outage is nominal. The outage time depends on the size of cache to flush, the number of datafiles, and the time required to break the mirror.

Note the following restrictions for the SUSPEND/RESUME feature:

- In a RAC configuration, you should not start a new instance while the original nodes are suspended.
- No checkpoint is initiated by the ALTER SYSTEM SUSPEND or ALTER SYSTEM RESUME statements.
- You cannot issue SHUTDOWN with IMMEDIATE, NORMAL, or TRANSACTIONAL options while the database is suspended.
- Issuing SHUTDOWN ABORT on a database that was already suspended reactivates the database. This prevents media recovery or crash recovery from hanging.

To make a split mirror backup in SUSPEND mode:

1. Place the database tablespaces in backup mode. For example, to place tablespace users in backup mode enter:

   ```
   ALTER TABLESPACE users BEGIN BACKUP;
   ```

   If you are backing up all of the tablespaces for your database, you can instead use:

   ```
   ALTER DATABASE BEGIN BACKUP;
   ```

2. If your mirror system has problems with splitting a mirror while disk writes are occurring, then suspend the database. For example, issue the following:

   ```
   ALTER SYSTEM SUSPEND;
   ```

3. Check to make sure that the database is suspended by querying V$INSTANCE. For example:

   ```
   SELECT DATABASE_STATUS FROM V$INSTANCE;

   DATABASE_STATUS
   -----------------
   SUSPENDED
   ```

4. Split the mirrors at the operating system or hardware level.

5. End the database suspension. For example, issue the following statement:

```
ALTER SYSTEM RESUME;
```

6. Check to make sure that the database is active by querying V$INSTANCE. For example, enter:

```
SELECT DATABASE_STATUS FROM V$INSTANCE;

DATABASE_STATUS
-----------------
ACTIVE
```

7. Take the specified tablespaces out of backup mode. For example, enter the following to take tablespace users out of backup mode:

```
ALTER TABLESPACE users END BACKUP;
```

8. Copy the control file and archive the online redo logs as usual for a backup.

Caution: Do not use the ALTER SYSTEM SUSPEND statement as a substitute for placing a tablespace in backup mode.

See Also: *Oracle Database Administrator's Guide* for more information about the SUSPEND/RESUME feature, and *Oracle Database SQL Reference* for more information about the ALTER SYSTEM statement

Making User-Managed Backups to Raw Devices

A **raw device** is a disk or partition that does not have a file system. In other words, a raw device can contain only a single file. Backing up files on raw devices poses operating system specific issues. The following sections discuss some of these issues on two of the most common operating systems supporting Oracle: UNIX and Windows.

See Also: *Oracle Real Application Clusters Installation and Configuration Guide* for a general overview of raw devices as they relate to Oracle Real Application Clusters

Backing Up to Raw Devices on UNIX

When backing up to or from raw devices, the UNIX `dd` command is the most common backup utility. See your operating system specific documentation for complete details about this utility.

The most important aspect of using `dd` is determining which options to specify. You need to know the following information.

Data	Explanation
Block size	You can specify the size of the buffer that `dd` uses to copy data. For example, you can specify that `dd` should copy data in units of 8 KB or 64 KB. Note that the block size for `dd` need not correspond to either the Oracle block size or the operating system block size: it is merely the size of the buffer used by `dd` when making the copy.
Raw offset	On some systems, the beginning of the file on the raw device is reserved for use by the operating system. This storage space is called the **raw offset**. Oracle should not back up or restore these bytes.
Size of Oracle block `0`	At the beginning of every Oracle file, the operating system-specific code places an Oracle block called **block 0**. The generic Oracle code does not recognize this block, but the block is included in the size of the file on the operating system. Typically, this block is the same size as the other Oracle blocks in the file.

The information in the preceding table enables you to set the `dd` options specified in Table 16–1.

Table 16–1 Options for dd Command

This option ...	Specifies ...
`if`	The name of the input file, that is, the file that you are reading.
`of`	The name of the output file, that is, the file to which you are writing.
`bs`	The buffer size used by `dd` to copy data.
`skip`	The number of `dd` buffers to skip on the input raw device if a raw offset exists. For example, if you are backing up a file on a raw device with a 64 KB raw offset, and the `dd` buffer size is 8 KB, then you can specify `skip=8` so that the copy starts at offset 64 KB.

Table 16–1 Options for dd Command

This option ...	Specifies ...
`seek`	The number of `dd` buffers to skip on the output raw device if a raw offset exists. For example, if you are backing up a file onto a raw device with a 64 KB raw offset, and the `dd` buffer size is 8 KB, then you can specify `skip=8` so that the copy starts at offset 64 KB.
`count`	The number of blocks on the input raw device for `dd` to copy. It is best to specify the exact number of blocks to copy when copying from raw device to file system, otherwise any extra space at the end of the raw volume that is not used by the Oracle datafile is copied to the file system. Remember to include block `0` in the total size of the input file. For example, if the `dd` block size is 8 KB, and you are backing up a 30720 KB datafile, then you can set `count=3841`. This value for `count` actually backs up 30728 KB: the extra 8 KB are for Oracle block `0`.

Because a raw device can be the input or output device for a backup, you have four possible scenarios for the backup. The possible options for `dd` depend on which scenario you choose, as illustrated in Table 16–2.

Table 16–2 Scenarios Involving dd Backups

Backing Up from ...	Backing Up to ...	Options Specified for dd Command
Raw device	Raw device	`if`, `of`, `bs`, `skip`, `seek`, `count`
Raw device	File system	`if`, `of`, `bs`, `skip`, `count`
File system	Raw device	`if`, `of`, `bs`, `seek`
File system	File system	`if`, `of`, `bs`

Backing Up with the dd utility on UNIX: Examples

For these examples of `dd` utility usage, assume the following:

- You are backing up a 30720 KB datafile.
- The beginning of the datafile has a block 0 of 8 KB.
- The raw offset is 64 KB.
- You set the `dd` block size to 8 KB when a raw device is involved in the copy.

In the following example, you back up from one raw device to another raw device:

```
% dd if=/dev/rsd1b of=/dev/rsd2b bs=8k skip=8 seek=8 count=3841
```

In the following example, you back up from a raw device to a file system:

```
% dd if=/dev/rsd1b of=/backup/df1.dbf bs=8k skip=8 count=3841
```

In the following example, you back up from a file system to a raw device:

```
% dd if=/backup/df1.dbf of=/dev/rsd2b bs=8k seek=8
```

In the following example, you back up from a file system to a file system, and so can set the block size to a high value to boost I/O performance:

```
% dd if=/oracle/dbs/df1.dbf of=/backup/df1.dbf bs=1024k
```

Backing Up to Raw Devices on Windows

Like UNIX, Windows supports raw disk partitions in which the database can store datafiles, online logs, and control files. Each raw partition is assigned either a drive letter or physical drive number and does not contain a file system. As in UNIX, each raw partition on NT is mapped to a single file.

NT differs from UNIX in the naming convention for Oracle files. On NT, raw datafile names are formatted as follows:

```
\\.\drive_letter:
\\.\PHYSICALDRIVEdrive_number
```

For example, the following are possible raw filenames:

```
\\.\G:
\\.\PHYSICALDRIVE3
```

Note that you can also create aliases to raw filenames. The standard Oracle database installation provides a `SETLINKS` utility that can create aliases such as `\\.\Datafile12` that point to filenames such as `\\.\PHYSICALDRIVE3`.

The procedure for making user-managed backups of raw datafiles is basically the same as for copying files on an NT file system, except that you should use the Oracle `OCOPY` utility rather than the NT-supplied `copy.exe` or `ntbackup.exe` utilities. `OCOPY` supports 64-bit file I/O, physical raw drives, and raw files. Note that `OCOPY` cannot back up directly to tape.

To display online documentation for `OCOPY`, enter `OCOPY` by itself at the Windows prompt. Sample output follows:

```
Usage of OCOPY:
     ocopy from_file [to_file [a | size_1 [size_n]]]
     ocopy -b from_file to_drive
```

```
ocopy -r from_drive to_dir
```

Note the important OCOPY options described in the following table.

This option ...	Specifies ...
b	Splits the input file into multiple output files. This option is useful for backing up to devices that are smaller than the input file.
r	Combines multiple input files and writes to a single output file. This option is useful for restoring backups created with the -b option.

Backing Up with OCOPY: Example

In this example, assume the following:

- Datafile 12 is mounted on the \\.\G: raw partition.
- The C: drive mounts a file system.
- The database is open.

To back up the datafile on the raw partition \\.\G: to a local file system, you can run the following command at the prompt after placing datafile 12 in backup mode:

```
OCOPY "\\.G:" C:\backup\datafile12.bak
```

Specifying the -b and -r Options for OCOPY: Example

In this example, assume the following:

- \\.\G: is a raw partition containing datafile 7
- The A: drive is a removable disk drive.
- The database is open.

To back up the datafile onto drive A:, you can execute the following command at the NT prompt after placing datafile 7 in backup mode:

```
# first argument is filename, second argument is drive
OCOPY -b "\\.\G:" A:\
```

When drive A: fills up, you can use another disk. In this way, you can divide the backup of datafile 7 into multiple files.

Similarly, to restore the backup, take the tablespace containing datafile 7 offline and run this command:

```
# first argument is drive, second argument is directory
```

```
OCOPY -r A:\ "\\.\G:"
```

Verifying User-Managed Backups

You should periodically verify your backups to ensure that they are usable for recovery. This section contains the following topics:

- Testing the Restore of Backups
- Running the DBVERIFY Utility

Testing the Restore of Backups

The best way to test the usability of backups is to restore them to a separate host and attempt to open the database, performing media recovery if necessary. This option requires that you have a separate host available for the restore procedure.

See Also:

- "Restoring Datafiles with Operating System Utilities" on page 17-6
- "Restoring Control Files" on page 17-8
- "Restoring Archived Redo Logs with Operating System Utilities" on page 17-6
- "Performing Complete User-Managed Media Recovery" on page 17-21 to learn how to recover files with SQL*Plus

Running the DBVERIFY Utility

The DBVERIFY program is an external command-line utility that performs a physical data structure integrity check on an offline datafile. Use DBVERIFY primarily when you need to ensure that a user-managed backup of a datafile is valid before it is restored or as a diagnostic aid when you have encountered data corruption problems.

The name and location of DBVERIFY is dependent on your operating system. For example, to perform an integrity check on datafile `tbs_52.f` on UNIX, you can run the `dbv` command as follows:

```
% dbv file=tbs_52.f
```

Sample dbv output follows:

```
DBVERIFY - Verification starting : FILE = users01.dbf


DBVERIFY - Verification complete

Total Pages Examined         : 250
Total Pages Processed (Data) : 1
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 0
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 2
Total Pages Processed (Seg)  : 0
Total Pages Failing   (Seg)  : 0
Total Pages Empty            : 247
Total Pages Marked Corrupt   : 0
Total Pages Influx           : 0
```

See Also: *Oracle Database Utilities* to learn about DBVERIFY

Making Logical Backups with Oracle Export Utilities

Oracle import and export utilities move Oracle data in and out of Oracle databases. Export utilities write exported database objects to operating system files in an Oracle-proprietary format. Import utilities can read the files produced by export utilities and re-create database objects. Logical exports of data can be a useful supplement to physical database backups in some situations, especially in backing up recovery catalog databases.

There are two sets of Oracle database import and export utilities: Original Import and Export (which were used in previous releases) and Data Pump Import and Export (new for Oracle Database Release 10*g)*. The Data Pump utilities offer better performance and more complete support of features of Oracle Database Release 10*g*. Whichever export tool you use to export objects to a file, you must use the corresponding import tool to import the objects from the file, that is, you cannot use the Data Pump Import utility to read the output of the Original Export utility, or the Original Import to read the output of the Data Pump Export utility.

See Also: *Oracle Database Utilities* for complete documentation of the Oracle import and export utilities, including a comparison of their capabilities.

Making User-Managed Backups of Miscellaneous Oracle Files

Always back up initialization parameter files, networking and configuration files, and password files. If a media failure destroys these files, then you may have difficulty re-creating your environment. For example, if you back up the database and server parameter file but do not back up the networking files, then you can restore and recover the database but will not be able to authenticate users through Oracle Net until you re-create the networking files.

As a rule, you should back up miscellaneous Oracle files after changing them. For example, if you add or change the net service names that can be used to access the database, then create a new backup of the `tnsnames.ora` file.

The easiest way to find configuration files is to start in the Oracle home directory and do a recursive search for all files ending in the `.ora` extension. For example, on UNIX you can run this command:

```
% find $ORACLE_HOME -name "*.ora" -print
```

You must use third-party utilities to back up the configuration files. For example, you can use the UNIX cp command to back up the `tnsnames.ora` and `listener.ora` files as follows:

```
% cp $ORACLE_HOME/network/admin/tnsnames.ora /d2/tnsnames‘date "+%m_%d_%y"‘.ora
% cp $ORACLE_HOME/network/admin/listener.ora /d2/listener‘date "+%m_%d_%y"‘.ora
```

You can also use an operating system utility to back up the server parameter file. Although the database does not depend on the existence of a particular version of the server parameter file to be started, you should keep relatively current backups of this file so that you do not lose changes made to the file. Note that if you lose the server parameter file, you can always create a new one or start the instance with a client-side initialization parameter file (`PFILE`).

Keeping Records of Current and Backup Database Files

One of the most important aspects of user-managed backup and recovery is keeping records of all current database files as well as the backups of these files. For example, you should have records for the location of the following files:

- Datafiles and control files
- Online and archived redo logs (note that online logs are never backed up)
- Initialization parameter files

- Password files
- Networking-related files

Recording the Locations of Datafiles, Control Files, and Online Redo Logs

The following useful SQL script displays the location of all control files, datafiles, and online redo log files for the database:

```
SELECT NAME FROM V$DATAFILE
UNION ALL
SELECT MEMBER FROM V$LOGFILE
UNION ALL
SELECT NAME FROM V$CONTROLFILE;
```

> **See Also:** *Oracle Database Reference* for more information on the V$ views

Recording the Locations of Archived Redo Logs

You can determine the location of the default archived log destinations by executing the following SQL script:

```
SELECT NAME, VALUE
FROM V$PARAMETER
WHERE NAME LIKE log_archive_dest%
AND VALUE IS NOT NULL
/

NAME                             VALUE
-------------------------------- ----------------------------------------
log_archive_dest_1               LOCATION=/oracle/work/arc_dest/arc
log_archive_dest_state_1         enable
```

Determine the format for archived logs by running SHOW as follows:

```
SHOW PARAMETER LOG_ARCHIVE_FORMAT
```

To see a list of all the archived logs recorded in the control file, issue this query:

```
SELECT NAME FROM V$ARCHIVED_LOG;
```

Recording the Locations and Dates of Backup Files

It is not enough to merely record the location of backup files: you must correlate the backups with the original files. If possible, name the backups with the same relative

filename as the primary file. Whatever naming system you use, keep a table containing the relevant information. For example, you could keep the following table as a record of database file locations in case of a restore emergency.

Datafile #	Tbs	Current Datafiles	Backup Datafiles
0 (cf)	0 (cf)	/oracle/oradata/trgt/control01.dbf	/d2/control01_10_31_02.dbf
1	SYSTEM	/oracle/oradata/trgt/system01.dbf	/d2/system01_10_31_02.dbf
2	undo	/oracle/oradata/trgt/undo01.dbf	/d2/undo01_10_31_02.dbf
3	cwmlite	/oracle/oradata/trgt/cwmlite01.dbf	/d2/cwmlite01_10_31_02.dbf
4	drsys	/oracle/oradata/trgt/drsys01.dbf	/d2/drsys01_10_31_02.dbf

17

Performing User-Managed Database Flashback and Recovery

This chapter describes how to restore and recover a database. It includes the following topics:

User-Managed Backup and Flashback Features of Oracle

Oracle's flashback features, which let you undo damage to your database after logical data corruption, include the following:

- Oracle Flashback Database, which returns your entire database to a previous state without requiring you to restore files from backup;
- Oracle Flashback Table, which returns one or more tables to their contents at a previous time;
- Oracle Flashback Drop, which undoes the effects of the DROP TABLE operation;
- Oracle Flashback Query, which allows you to query the contents of the database at a past time;
- Oracle Flashback Version Query, which lets you view past states of data;
- Oracle Flashback Transaction Query, which allows you review transactions affecting a table over time.

All of these operations are available within SQL*Plus, and none of them require the use of Recovery Manager. More details about using the flashback features of Oracle in data recovery situations are provided in "Oracle Flashback Technology: Overview" on page 9-2.

Performing Flashback Database with SQL*Plus

The SQL*Plus FLASHBACK DATABASE command performs the same function as the RMAN FLASHBACK DATABASE command: it returns the database to a prior state.

Note that using Flashback Database requires that you create a flash recovery area for your database and enable the collection of flashback logs. See "Oracle Flashback Database: Alternative to Point-In-Time Recovery" on page 9-15 for more details about how the Flashback Database feature works, requirements for using Flashback Database , and how to enable collection of flashback logs required for Flashback Database. The requirements and preparations are the same whether you use RMAN or user-managed backup and recovery.

To perform the FLASHBACK DATABASE operation:

1. Query the target database to determine the range of possible flashback SCNs. The following SQL*Plus queries show you the the latest and earliest SCN in the flashback window:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE;

SQL> SELECT OLDEST_FLASHBACK_SCN, OLDEST_FLASHBACK_TIME
     FROM V$FLASHBACK_DATABASE_LOG;
```

2. Use other flashback features if necessary, to identify the SCN or time of the unwanted changes to your database.
3. Start SQL*Plus with administrator privileges, and run the FLASHBACK DATABASE statement to return the database to a prior TIMESTAMP or SCN. For example:

```
FLASHBACK DATABASE TO SCN 46963;
FLASHBACK DATABASE TO TIMESTAMP (SYSDATE-1/24);
FLASHBACK DATABASE TO TIMESTAMP timestamp'2002-11-05 14:00:00';
FLASHBACK DATABASE
  TO TIMESTAMP to_timestamp('2002-11-11 16:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

   Open the database read-only to examine the results of the Flashback Database operation. When the operation completes, you can open the database read-only and perform some queries to make sure you have recovered the data you need. If you find that you need to perform Flashback Database again to a different target time, then use RECOVER DATABASE to return the database back to the present time, and then try another FLASHBACK DATABASE statement.

   If you are satisfied with the results of Flashback Database, then you can re-open your database with the RESETLOGS option. If appropriate, you can also use an Oracle export utililty like Data Pump Export to save lost data, use RECOVER DATABASE to return the database to the present, and re-import the lost object.

About User-Managed Restore Operations

To restore a file is to replace it with a backup file. Typically, you restore a file when a media failure or user error has damaged or deleted the original file. The following files are candidates for restore operations:

- Datafiles and control files
- Archived redo logs
- Server parameter file

In each case, the loss of a primary file and the restore of a backup has the following implications for media recovery.

If you lose . . .	Then . . .
One or more datafiles	You must restore them from a backup and perform media recovery. Recovery is required whenever the checkpoint SCN in the datafile header does not match the checkpoint SCN for the datafile that is recorded in the control file.
All copies of the current control file	You must restore a backup control file and then open the database with the RESETLOGS option. If you do not have a backup, then you can attempt to re-create the control file. If possible, use the script included in the ALTER DATABASE BACKUP CONTROLFILE TO TRACE output. Additional work may be required to match the control file structure with the current database structure.
One copy of a multiplexed control file	Copy one of the intact multiplexed control files into the location of the damaged or missing control file and open the database. If you cannot copy the control file to its original location, then edit the initialization parameter file to reflect a new location or remove the damaged control file. Then, open the database.
One or more archived logs required for media recovery	You must restore backups of these archived logs for recovery to proceed. You can restore either to the default or nondefault location. If you do not have backups, then you must performing incomplete recovery up to an SCN before the first missing redo log and open RESETLOGS.
The server parameter file	If you have a backup of the server parameter file, then restore it. Alternatively, if you have a backup of the client-side initialization parameter file, then you can restore a backup of this file, start the instance, and then re-create the server parameter file.

Note: Restore and recovery of Oracle-managed files is no different from restore and recovery of user-named files.

Determining Which Datafiles Require Recovery

You can use the dynamic performance view V$RECOVER_FILE to determine which files to restore in preparation for media recovery. This view lists all files that need to be recovered and explains why they need to be recovered.

The following query displays the file ID numbers of datafiles that require media recovery as well as the reason for recovery (if known) and the SCN and time when recovery needs to begin:

```
SELECT * FROM V$RECOVER_FILE;
```

```
FILE#      ONLINE  ERROR              CHANGE#    TIME
---------- ------- ----------------- ---------- ---------
        14 ONLINE                             0
        15 ONLINE  FILE NOT FOUND             0
        21 OFFLINE OFFLINE NORMAL             0
```

> **Note:** The view is not useful if the control file currently in use is a restored backup or a new control file created after the media failure occurred. A restored or re-created control file does not contain the information the database needs to populate V$RECOVER_FILE accurately.

Query V$DATAFILE and V$TABLESPACE to obtain filenames and tablespace names for datafiles requiring recovery. For example, enter:

```
SELECT d.NAME, t.NAME AS tablespace_name
FROM V$DATAFILE d, V$TABLESPACE t
WHERE t.TS# = d.TS#
AND d.FILE# IN (14,15,21);  # use values obtained from V$RECOVER_FILE query
```

You can combine these queries in the following SQL*Plus script (sample output show in the following example):

```
COL df# FORMAT 999
COL df_name FORMAT a20
COL tbsp_name FORMAT a10
COL status FORMAT a7
COL error FORMAT a10
SELECT r.FILE# AS df#, d.NAME AS df_name, t.NAME AS tbsp_name,
       d.STATUS, r.ERROR, r.CHANGE#, r.TIME
FROM V$RECOVER_FILE r, V$DATAFILE d, V$TABLESPACE t
WHERE t.TS# = d.TS#
AND d.FILE# = r.FILE#
/
```

Restoring Datafiles and Archived Redo Logs

This section contains the following topics:

- Restoring Datafiles with Operating System Utilities
- Restoring Archived Redo Logs with Operating System Utilities

Restoring Datafiles with Operating System Utilities

If a media failure permanently damages one or more datafiles of a database, then you must restore backups of these datafiles before you can recover the damaged files. If you cannot restore a damaged datafile to its original location (for example, you must replace a disk, so you restore the files to an alternate disk), then you must indicate the new locations of these files to the control file.

If you are restoring a database file on a raw disk or partition, then the procedure is basically the same as when restoring to a file on a file system. However, be aware of the naming conventions for files on raw devices (which differ depending on the operating system), and use an operating system utility that supports raw devices.

> **See Also:** "Making User-Managed Backups to Raw Devices" on page 16-20 for an overview of considerations when backing up and restoring files on raw devices

To restore backup datafiles to their default location:

1. Determine which datafiles to recover by using the techniques described in "Determining Which Datafiles Require Recovery" on page 17-4.
2. If the database is open, then take the tablespaces containing the inaccessible datafiles offline. For example, enter:

   ```
   ALTER TABLESPACE users OFFLINE IMMEDIATE;
   ```

3. Copy backups of the damaged datafiles to their default location using operating system commands. For example, to restore `users01.dbf` you might issue:

   ```
   % cp /disk2/backup/users01.dbf $ORACLE_HOME/oradata/trgt/users01.dbf
   ```

4. Recover the affected tablespace. For example, enter:

   ```
   RECOVER TABLESPACE users
   ```

5. Bring the recovered tablespace online. For example, enter:

   ```
   ALTER TABLESPACE users ONLINE;
   ```

Restoring Archived Redo Logs with Operating System Utilities

All archived redo logs generated between the time a restored backup was created and the target recovery time are required for the pending recovery. The archived logs will eventually need to be on disk so that they are available to the database.

To restore necessary archived redo logs:

1. To determine which archived redo log files are needed, query V$ARCHIVED_LOG and V$RECOVERY_LOG. If a datafile requires recovery, but no backup of the datafile exists, then you need all redo generated starting from the time when the datafile was added to the database.

View	Description
V$ARCHIVED_LOG	Lists filenames for all the archived logs.
V$RECOVERY_LOG	Lists only the archived redo logs that the database needs to perform media recovery. It also includes the probable names of the files, using LOG_ARCHIVE_FORMAT. **Note:** This view is only populated when recovery is required for a datafile. Hence, this view is not useful in the case of a planned recovery such as a user error.

2. If space is available, then restore the required archived redo log files to the location specified by LOG_ARCHIVE_DEST_1. The database locates the correct log automatically when required during media recovery. For example, enter:

```
% cp /disk2/arch/* $ORACLE_HOME/oradata/trgt/arch
```

3. If sufficient space is not available at the location indicated by the archiving destination initialization parameter, restore some or all of the required archived redo log files to an alternate location. Specify the location before or during media recovery using the LOGSOURCE parameter of the SET statement in SQL*Plus or the RECOVER . . . FROM parameter of the ALTER DATABASE statement in SQL. For example, enter:

```
SET LOGSOURCE /tmp   # set location using SET statement
  DATABASE RECOVER FROM '/tmp';  # set location in RECOVER statement
```

4. After an archived log is applied, and after making sure that a copy of each archived log group still exists in offline storage, delete the restored copy of the archived redo log file to free disk space. For example:

```
% rm /tmp/*.dbf
```

> **See Also:** *Oracle Database Reference* for more information about the data dictionary views, and "About User-Managed Media Recovery" on page 17-14 for an overview of log application during media recovery

Restoring Control Files

This section contains the following topics:

Losing a Member of a Multiplexed Control File

Use the following procedures to recover a database if a permanent media failure has damaged one or more control files of a database and at least one control file has *not* been damaged by the media failure.

Copying a Multiplexed Control File to a Default Location

If the disk and file system containing the lost control file are intact, then you can simply copy one of the intact control files to the location of the missing control file. In this case, you do not have to alter the `CONTROL_FILES` initialization parameter setting.

To replace a damaged control file by copying a multiplexed control file:

1. If the instance is still running, then shut it down:

   ```
   SHUTDOWN ABORT
   ```

2. Correct the hardware problem that caused the media failure. If you cannot repair the hardware problem quickly, then proceed with database recovery by restoring damaged control files to an alternative storage device, as described in "Copying a Multiplexed Control File to a Nondefault Location" on page 17-9.

3. Use an intact multiplexed copy of the database's current control file to copy over the damaged control files. For example, to replace `bad_cf.f` with `good_cf.f`, you might enter:

   ```
   % cp /oracle/good_cf.f /oracle/dbs/bad_cf.f
   ```

4. Start a new instance and mount and open the database. For example, enter:

   ```
   STARTUP
   ```

Copying a Multiplexed Control File to a Nondefault Location

Assuming that the disk and file system containing the lost control file are not intact, then you cannot copy one of the good control files to the location of the missing control file. In this case, you must alter the CONTROL_FILES initialization parameter to indicate a new location for the missing control file.

To restore a control file to a nondefault location:

1. If the instance is still running, then shut it down:

   ```
   SHUTDOWN ABORT
   ```

2. If you cannot correct the hardware problem that caused the media failure, then copy the intact control file to alternative locations. For example, to copy a good version of control01.dbf to a new disk location you might issue:

   ```
   % cp $ORACLE_HOME/oradata/trgt/control01.dbf /new_disk/control01.dbf
   ```

3. Edit the parameter file of the database so that the CONTROL_FILES parameter reflects the current locations of all control files and excludes all control files that were not restored. Assume the initialization parameter file contains:

   ```
   CONTROL_FILES='/oracle/oradata/trgt/control01.dbf','/bad_disk/control02.dbf'
   ```

   Then, you can edit it as follows:

   ```
   CONTROL_FILES='/oracle/oradata/trgt/control01.dbf','/new_disk/control02.dbf'
   ```

4. Start a new instance and mount and open the database. For example:

   ```
   STARTUP
   ```

Losing All Current Control Files When a Backup Is Available

Use the following procedures to restore a backup control file if a permanent media failure has damaged all control files of a database and you have a backup of the control file. When a control file is inaccessible, you can start the instance, but not mount the database. If you attempt to mount the database when the control file is unavailable, then you receive this error message:

```
ORA-00205: error in identifying controlfile, check alert log for more info
```

You cannot mount and open the database until the control file is accessible again. If you restore a backup control file, then you must open RESETLOGS.

As indicated in Table 17–1, the procedure for restoring the control file depends on whether the online redo logs are available.

Table 17–1 Scenarios When Control Files Are Lost

Status of Online Logs	Status of Datafiles	Response
Available	Current	If the online logs contain redo necessary for recovery, then restore a backup control file and apply the logs during recovery. You must specify the filename of the online logs containing the changes in order to open the database. After recovery, open RESETLOGS.
Unavailable	Current	If the online logs contain redo necessary for recovery, then re-create the control file. Because the online redo logs are inaccessible, open RESETLOGS (when the online logs are accessible it is not necessary to OPEN RESETLOGS after recovery with a created control file).
Available	Backup	Restore a backup control file, perform complete recovery, and then open RESETLOGS.
Unavailable	Backup	Restore a backup control file, perform incomplete recovery, and then open RESETLOGS.

Restoring a Backup Control File to the Default Location

If possible, restore the control file to its original location. In this way, you avoid having to specify new control file locations in the initialization parameter file.

To restore a backup control file to its default location:

1. If the instance is still running, shut it down:

   ```
   SHUTDOWN ABORT
   ```

2. Correct the hardware problem that caused the media failure.

3. Restore the backup control file to all locations specified in the CONTROL_FILES parameter. For example, if *ORACLE_HOME*/oradata/trgt/control01.dbf and *ORACLE_HOME*/oradata/trgt/control02.dbf are the control file locations listed in the server parameter file, then use an operating system utility to restore the backup control file to these locations:

   ```
   % cp /backup/control01.dbf ORACLE_HOME/oradata/trgt/control01.dbf
   % cp /backup/control02.dbf ORACLE_HOME/oradata/trgt/control02.dbf
   ```

4. Start a new instance and mount the database. For example, enter:

   ```
   STARTUP MOUNT
   ```

5. Begin recovery by executing the RECOVER command with the USING BACKUP CONTROLFILE clause. Specify UNTIL CANCEL if you are performing incomplete recovery. For example, enter:

   ```
   RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL
   ```

6. Apply the prompted archived logs. If you then receive another message saying that the required archived log is missing, it probably means that a necessary redo record is located in the online redo logs. This situation can occur when unarchived changes were located in the online logs when the instance crashed.

   For example, assume that you see the following:

   ```
   ORA-00279: change 55636 generated at 11/08/2002 16:59:47 needed for thread 1
   ORA-00289: suggestion : /oracle/work/arc_dest/arcr_1_111.arc
   ORA-00280: change 55636 for thread 1 is in sequence #111
   Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
   ```

   You can specify the name of an online redo log and press Enter (you may have to try this a few times until you find the correct log):

   ```
   ORACLE_HOME/oradata/redo01.dbf
   Log applied.
   Media recovery complete.
   ```

   If the online logs are inaccessible, then you can cancel recovery without applying them. If all datafiles are current, and if redo in the online logs is required for recovery, then you cannot open the database without applying the online logs. If the online logs are inaccessible, then you must re-create the control file (refer to "Losing All Current and Backup Control Files" on page 17-12).

7. Open the database with the RESETLOGS option after finishing recovery:

   ```
   ALTER DATABASE OPEN RESETLOGS;
   ```

Restoring a Backup Control File to a Nondefault Location

If you cannot restore the control file to its original place because the media damage is too severe, then you must specify new control file locations in the server parameter file. A valid control file must be available in all locations specified by the

CONTROL_FILES initialization parameter. If not, then the database prevents you from the mounting the database.

To restore a control file to a nondefault location:

Follow the steps in "Restoring a Backup Control File to the Default Location" on page 17-10, except after step 2 add the following step:

Edit all locations specified in the CONTROL_FILES initialization parameter to reflect the new control file locations. For example, if the control file locations listed in the server parameter file are as follows, and both locations are inaccessible:

```
CONTROL_FILES='/oracle/oradata/trgt/control01.dbf',
              '/oracle/oradata/trgt/control01.dbf'
```

Then, you can edit the initialization parameter file as follows:

```
CONTROL_FILES='/good_disk/control01.dbf','/good_disk/control02.dbf'
```

Losing All Current and Backup Control Files

If all control files have been lost in a permanent media failure, but all online redo log members remain intact, then you can recover the database after creating a new control file. The advantage of this tactic is that you are *not* required to open the database with the RESETLOGS option.

Depending on the existence and currency of a control file backup, you have the options listed in Table 17–2 for generating the text of the CREATE CONTROLFILE statement. Note that changes to the database are recorded in the alert_*SID*.log, so check this log when deciding which option to choose.

Table 17–2 Options for Creating the Control File **(Page 1 of 2)**

If you . . .	Then . . .
Executed ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS after you made the last structural change to the database, and if you have saved the SQL command trace output	Use the CREATE CONTROLFILE statement from the trace output as-is.
Performed your most recent execution of ALTER DATABASE BACKUP CONTROLFILE TO TRACE before you made a structural change to the database	Edit the output of ALTER DATABASE BACKUP CONTROLFILE TO TRACE to reflect the change. For example, if you recently added a datafile to the database, then add this datafile to the DATAFILE clause of the CREATE CONTROLFILE statement.

Table 17–2 Options for Creating the Control File **(Page 2 of 2)**

If you . . .	Then . . .
Backed up the control file with the ALTER DATABASE BACKUP CONTROLFILE TO *filename* statement (not the TO TRACE option)	Use the control file copy to obtain SQL output. Create a temporary database instance, mount the backup control file, and then run ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS. If the control file copy predated a recent structural change, then edit the trace to reflect the change.
Do not have a control file backup in either TO TRACE format or TO *filename* format	Execute the CREATE CONTROLFILE statement manually (refer to *Oracle Database SQL Reference*).

Note: If your character set is not the default US7ASCII, then you must specify the character set as an argument to the CREATE CONTROLFILE statement. The database character set is written to the alert log at startup. The character set information is also recorded in the BACKUP CONTROLFILE TO TRACE output.

To create a new control file:

1. Start the database in NOMOUNT mode. For example, enter:

```
STARTUP NOMOUNT
```

2. Create the control file with the CREATE CONTROLFILE statement, specifying the NORESETLOGS option (refer to Table 17–2 for options). The following example assumes that the character set is the default US7ASCII:

```
CREATE CONTROLFILE REUSE DATABASE SALES NORESETLOGS ARCHIVELOG
     MAXLOGFILES 32
     MAXLOGMEMBERS 2
     MAXDATAFILES 32
     MAXINSTANCES 16
     MAXLOGHISTORY 1600
LOGFILE
     GROUP 1 (
       '/diska/prod/sales/db/log1t1.dbf',
       '/diskb/prod/sales/db/log1t2.dbf'
     )  SIZE 100K
     GROUP 2 (
       '/diska/prod/sales/db/log2t1.dbf',
       '/diskb/prod/sales/db/log2t2.dbf'
```

```
    )  SIZE 100K,
DATAFILE
    '/diska/prod/sales/db/database1.dbf',
    '/diskb/prod/sales/db/filea.dbf';
```

After creating the control file, the instance mounts the database.

3. Recover the database as normal (*without* specifying the USING BACKUP CONTROLFILE clause):

```
RECOVER DATABASE
```

4. Open the database after recovery completes (RESETLOGS option not required):

```
ALTER DATABASE OPEN;
```

5. Immediately back up the control file. The following SQL statement backs up a database's control file to /backup/control01.dbf:

```
ALTER DATABASE BACKUP CONTROLFILE TO '/backup/control01.dbf' REUSE;
```

> **See Also:** "Backing Up the Control File to a Trace File" on page 16-14, and "Recovering Through RESETLOGS with Created Control File: Scenario" on page 18-5

About User-Managed Media Recovery

During complete or incomplete media recovery, the database applies redo log files to the datafiles during the roll forward phase of media recovery. Because changes to undo segments are recorded in the online redo log, rolling forward regenerates the corresponding undo segments. Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time.

To perform recovery, Oracle Corporation recommends that you use the RECOVER SQL statement in SQL*Plus. You can also use the SQL statement ALTER DATABASE RECOVER, but the RECOVER statement is simpler in most cases.

Preconditions of Performing User-Managed Recovery

To start any type of media recovery, you must adhere to the following restrictions:

- You must have administrator privileges.
- All recovery sessions must be compatible.

- One session cannot start complete media recovery while another performs incomplete media recovery.
- You cannot start media recovery if you are connected to the database through a shared server process.

Applying Logs Automatically with the RECOVER Command

Oracle Corporation recommends that you use the SQL*Plus RECOVER command rather than the ALTER DATABASE RECOVER statement to perform media recovery. In almost all cases, the SQL*Plus method is easier.

When using SQL*Plus to perform media recovery, the easiest strategy is to perform automatic recovery. Automatic recovery initiates recovery without manually prompting SQL*Plus to apply each individual archived log.

When using SQL*Plus, you have two options for automating the application of the default filenames of archived redo logs needed during recovery:

- Issuing SET AUTORECOVERY ON before issuing the RECOVER command
- Specifying the AUTOMATIC keyword as an option of the RECOVER command

In either case, no interaction is required when you issue the RECOVER command if the necessary files are in the correct locations with the correct names. The filenames used when you use automatic recovery are derived from the concatenated values of LOG_ARCHIVE_FORMAT with LOG_ARCHIVE_DEST_*n*, where *n* is the highest value among all enabled, local destinations.

For example, assume the following initialization parameter settings are in effect in the database instance:

```
LOG_ARCHIVE_DEST_1 = "LOCATION=/arc_dest/loc1/"
LOG_ARCHIVE_DEST_2 = "LOCATION=/arc_dest/loc2/"
LOG_ARCHIVE_DEST_STATE_1 = DEFER
LOG_ARCHIVE_DEST_STATE_2 = ENABLE
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc
```

In this case, SQL*Plus automatically suggests the filename /arc_dest/loc2/arch_%t_%s.arc (where %t is the thread and %s is the sequence).

If you run SET AUTORECOVERY OFF, which is the default option, then you must enter the filenames manually, or accept the suggested default filename by pressing the Enter key.

Automating Recovery with SET AUTORECOVERY

Run the SET AUTORECOVERY ON command to enable on automatic recovery.

To automate the recovery using SET AUTORECOVERY:

1. Restore a backup of the offline datafiles. This example restores an inconsistent backup of all datafiles with an operating system utility:

   ```
   % cp /backup/datafiles/*.dbf $ORACLE_HOME/oradata/trgt/
   ```

2. Ensure the database is mounted. For example, if the database is shut down, run:

   ```
   STARTUP MOUNT
   ```

3. Enable automatic recovery. For example, in SQL*Plus run:

   ```
   SET AUTORECOVERY ON
   ```

4. Recover the desired datafiles. This example recovers the whole database:

   ```
   RECOVER DATABASE
   ```

   The database automatically suggests and applies the necessary archived logs.

5. Open the database. For example:

   ```
   ALTER DATABASE OPEN;
   ```

Note: After issuing the SQL*Plus RECOVER command, you can view all files that have been considered for recovery in the V$RECOVERY_FILE_STATUS view. You can access status information for each file in the V$RECOVERY_STATUS view. These views are not accessible after you terminate the recovery session.

Automating Recovery with the AUTOMATIC Option of the RECOVER Command

Besides using SET AUTORECOVERY to turn on automatic recovery, you can also simply specify the AUTOMATIC keyword in the RECOVER command.

To automate the recovery with the RECOVER AUTOMATIC command:

1. Restore a backup of the offline datafiles. This example restores a backup of all datafiles:

   ```
   % cp /backup/datafiles/*.dbf $ORACLE_HOME/oradata/trgt/
   ```

2. Ensure the database is mounted. For example, if the database is shut down, run:

```
STARTUP MOUNT
```

3. Recover the desired datafiles by specifying the AUTOMATIC keyword. This example performs automatic recovery on the whole database:

```
RECOVER AUTOMATIC DATABASE
```

   The database automatically suggests and applies the necessary archived logs.

4. Open the database. For example:

```
ALTER DATABASE OPEN;
```

If you use an Oracle Real Application Clusters configuration, and if you are performing incomplete recovery or using a backup control file, then the database can only compute the name of the first archived redo log file from the *first* redo thread. You may have to manually apply the first log file from the other redo threads. After the first log file in a given thread has been supplied, the database can suggest the names of the subsequent logs in this thread.

> **See Also:** Your operating system specific Oracle documentation for examples of log file application

Recovering When Archived Logs Are in the Default Location

Recovering when the archived logs are in their default location is the simplest case. As a log is needed, the database suggests the filename. If you are running nonautomatic media recovery with SQL*Plus, then the output is displayed in this format:

```
ORA-00279: Change #### generated at DD/MM/YY HH:MM:SS needed for thread#
ORA-00289: Suggestion : logfile
ORA-00280: Change #### for thread # is in sequence #
Specify log: [<RET> for suggested | AUTO | FROM logsource | CANCEL ]
```

For example, SQL*Plus displays output similar to the following:

```
ORA-00279: change 53577 generated at 11/26/02 19:20:58 needed for thread 1
ORA-00289: suggestion : /oracle/oradata/trgt/arch/arcr_1_802.arc
ORA-00280: change 53577 for thread 1 is in sequence #802
Specify log: [<RET> for suggested | AUTO | FROM logsource | CANCEL ]
```

Similar messages are returned when you use an ALTER DATABASE . . . RECOVER statement. However, no prompt is displayed.

The database constructs suggested archived log filenames by concatenating the current values of the initialization parameters LOG_ARCHIVE_DEST_*n* (where *n* is the highest value among all enabled, local destinations) and LOG_ARCHIVE_FORMAT and using log history data from the control file. The following are possible settings:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION = /oracle/oradata/trgt/arch/'
LOG_ARCHIVE_FORMAT = arcr_%t_%s.arc

SELECT NAME FROM V$ARCHIVED_LOG;

NAME
---------------------------------------
/oracle/oradata/trgt/arch/arcr_1_467.arc
/oracle/oradata/trgt/arch/arcr_1_468.arc
/oracle/oradata/trgt/arch/arcr_1_469.arc
```

Thus, if all the required archived log files are mounted at the LOG_ARCHIVE_DEST_1 destination, and if the value for LOG_ARCHIVE_FORMAT is never altered, then the database can suggest and apply log files to complete media recovery automatically.

Recovering When Archived Logs Are in a Nondefault Location

Performing media recovery when archived logs are not in their default location adds an extra step. You have the following mutually exclusive options:

- Edit the LOG_ARCHIVE_DEST_*n* parameter that specifies the location of the archived redo logs, then recover as usual.
- Use the SET statement in SQL*Plus to specify the nondefault log location before recovery, or the LOGFILE parameter of the RECOVER command

Resetting the Archived Log Destination

You can edit the initialization parameter file or issue ALTER SYSTEM statements to change the default location of the archived redo logs.

To change the default archived log location before recovery:

1. Use an operating system utility to restore the archived logs to a nondefault location. For example, enter:

   ```
   % cp /backup/arch/* /tmp/
   ```

2. Change the value for the archive log parameter to the nondefault location. You can issue ALTER SYSTEM statements while the instance is started, or edit the initialization parameter file and then start the database instance. For example, while the instance is shut down edit the parameter file as follows:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION=/tmp/'
LOG_ARCHIVE_FORMAT = arcr_%t_%s.arc
```

3. Using SQL*Plus, start a new instance by specifying the edited initialization parameter file, and then mount the database. For example, enter:

```
STARTUP MOUNT
```

4. Begin media recovery as usual. For example, enter:

```
RECOVER DATABASE
```

Overriding the Archived Log Destination

In some cases, you may want to override the current setting for the archiving destination parameter as a source for redo log files.

To recover archived logs in a nondefault location with SET LOGSOURCE:

1. Using an operating system utility, copy the archived redo logs to an alternative location. For example, enter:

```
% cp $ORACLE_HOME/oradata/trgt/arch/* /tmp
```

2. Specify the alternative location within SQL*Plus for the recovery operation. Use the LOGSOURCE parameter of the SET statement or the RECOVER . . . FROM clause of the ALTER DATABASE statement. For example, start SQL*Plus and run:

```
SET LOGSOURCE "/tmp"
```

3. Recover the offline tablespace. For example, to recover the offline tablespace users do the following:

```
RECOVER AUTOMATIC TABLESPACE users
```

4. Alternatively, you can avoid running SET LOGSOURCE and simply run:

```
RECOVER AUTOMATIC TABLESPACE users FROM "/tmp"
```

Note: Overriding the redo log source does not affect the archive redo log destination for online redo logs groups being archived.

Responding to Unsuccessful Application of Redo Logs

If you are using SQL*Plus's recovery options (not SQL statements), then each time the database successfully applies a redo log file, the following message is returned:

```
Log applied.
```

You are then prompted for the next log in the sequence or, if the most recently applied log is the last required log, terminates recovery.

If the suggested file is incorrect or you provide an incorrect filename, then the database returns an error message. For example, you may see something like:

```
ORA-00308: cannot open archived log "/oracle/oradata/trgt/arch/arcr_1_811.arc"
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

Recovery cannot continue until the required redo log is applied. If the database returns an error message after supplying a log filename, then the following responses are possible.

Error	Possible Cause	Solution
`ORA-27037: unable to obtain file status`	Entered wrong filename. Log is missing.	Reenter correct filename. Restore backup archived redo log.
`ORA-27047: unable to read the header block of file`	The log may have been partially written or become corrupted.	If you can locate an uncorrupted or complete log copy, then apply the intact copy and continue recovery. If no copy of the log exists and you know the time of the last valid redo entry, then you use incomplete recovery. Restore backups and restart recovery.

Interrupting User-Managed Media Recovery

If you start media recovery and must then interrupt it, for example, because a recovery operation must end for the night and resume the next morning, then take either of the following actions:

- Enter the word `CANCEL` when prompted for a redo log file.
- Use your operating system's interrupt signal if you must terminate when recovering an individual datafile, or when automated recovery is in progress.

After recovery is canceled, you can resume it later with the RECOVER command. Recovery resumes where it left off when it was canceled.

Performing Complete User-Managed Media Recovery

When you perform complete recovery, you recover the backups to the current SCN. You can either recover the whole database at once or recover individual tablespaces or datafiles. Because you do not have to open the database with the RESETLOGS option after complete recovery as you do after incomplete recovery, you have the option of recovering some datafiles at one time and the remaining datafiles later.

This section describes the steps necessary to complete media recovery operations, and includes the following topics:

- Performing Closed Database Recovery
- Performing Datafile Recovery in an Open Database

See Also: *Oracle Database Backup and Recovery Basics* for basic information about media recovery concepts, which apply in both user-managed and RMAN-based backup and recovery.

See Also: *Oracle Database Backup and Recovery Basics* to familiarize yourself with fundamental recovery concepts and strategies:

Performing Closed Database Recovery

This section describes steps to perform complete recovery while the database is not open. You can recover either all damaged datafiles in one operation, or perform individual recovery of each damaged datafile in separate operations.

Perform the media recovery in the following stages:

1. Prepare for closed database recovery as described in "Preparing for Closed Database Recovery" on page 17-22.
2. Restore the necessary files as described in "Restoring Backups of the Damaged or Missing Files" on page 17-22.
3. Recover the restored datafiles as described in "Recovering the Database" on page 17-23.

Preparing for Closed Database Recovery

In this stage, you shut down the instance and inspect the media device that is causing the problem.

To prepare for closed database recovery:

1. If the database is open, then shut it down. For example:

   ```
   SHUTDOWN IMMEDIATE
   ```

2. If you are recovering from a media error, then correct it if possible. If the hardware problem that caused the media failure was temporary, and if the data was undamaged (for example, a disk or controller power failure), then no media recovery is required: simply start the database and resume normal operations. If you cannot repair the problem, then proceed to the next stage.

Restoring Backups of the Damaged or Missing Files

In this stage, you restore all necessary backups.

To restore the necessary files:

1. Determine which datafiles to recover by using the techniques described in "Determining Which Datafiles Require Recovery" on page 17-4.

2. If the files are permanently damaged, then identify the most recent backups for the damaged files. Restore *only* the datafiles damaged by the media failure: do not restore any undamaged datafiles or any online redo log files.

   For example, if *ORACLE_HOME*`/oradata/trgt/users01.dbf` is the only damaged file, then you may determine that `/backup/users01_10_24_02.dbf` is the most recent backup of this file. If you do not have a backup of a specific datafile, then you may be able to create an empty replacement file that can be recovered.

3. Use an operating system utility to restore the files to their default location or to a new location. Restore the necessary files as described in "Restoring Datafiles and Archived Redo Logs" on page 17-5. For example, a UNIX user restoring `users01.dbf` to its default location might enter:

   ```
   % cp /backup/users01_10_24_02.dbf $ORACLE_HOME/oradata/trgt/users01.dbf
   ```

Use the following guidelines when determining where to restore datafile backups.

If . . .	Then . . .
The hardware problem is repaired and you can restore the datafiles to their default locations	Restore the datafiles to their default locations and begin media recovery.
The hardware problem persists and you cannot restore datafiles to their original locations	Restore the datafiles to an alternative storage device. Indicate the new location of these files in the control file with ALTER DATABASE RENAME FILE. Use the operation described in "Renaming and Relocating Datafiles" in the *Oracle Database Administrator's Guide*, as necessary.

Recovering the Database

In the final stage, you recover the datafiles that you have restored.

To recover the restored datafiles:

1. Connect to the database with administrator privileges, then start a new instance and mount, but do not open, the database. For example, enter:

   ```
   STARTUP MOUNT
   ```

2. Obtain the datafile names and statuses of all datafiles by checking the list of datafiles that normally accompanies the current control file or querying the V$DATAFILE view. For example, enter:

   ```
   SELECT NAME,STATUS FROM V$DATAFILE;
   ```

3. Ensure that all datafiles of the database are online. All datafiles of the database requiring recovery must be online unless an offline tablespace was taken offline normally or is part of a read-only tablespace. For example, to guarantee that a datafile named `/oracle/dbs/tbs_10.f` is online, enter the following:

   ```
   ALTER DATABASE DATAFILE '/oracle/dbs/tbs_10.f' ONLINE;
   ```

   If a specified datafile is already online, then the database ignores the statement. If you prefer, create a script to bring all datafiles online at once as in the following:

   ```
   SPOOL onlineall.sql
   SELECT 'ALTER DATABASE DATAFILE '''||name||''' ONLINE;' FROM V$DATAFILE;
   SPOOL OFF
   ```

```
SQL> @onlineall
```

4. Issue the statement to recover the database, tablespace, or datafile. For example, enter one of the following RECOVER command:

```
RECOVER DATABASE   # recovers whole database
RECOVER TABLESPACE users   # recovers specific tablespace
RECOVER DATAFILE '?/oradata/trgt/users01.dbf';   # recovers specific
datafile
```

Follow these guidelines when deciding which statement to execute:

If you want to . . .	Then . . .
Recover all damaged files in one step	Execute RECOVER DATABASE
Recover an individual tablespace	Execute RECOVER TABLESPACE
Recover an individual damaged datafile	Execute RECOVER DATAFILE
Parallelize recovery of the whole database or an individual datafile	Refer to "Performing Media Recovery in Parallel" on page 17-39

5. If you choose not to automate the application of archived logs, then you must accept or reject each prompted log. If you automate recovery, then the database applies the logs automatically. Recovery continues until all required archived and online redo logs have been applied to the restored datafiles.
6. The database notifies you when media recovery is complete:

```
Media recovery complete.
```

If no archived redo log files are required for complete media recovery, then the database applies all necessary online redo log files and terminates recovery.

7. After recovery terminates, open the database for use:

```
ALTER DATABASE OPEN;
```

> **See Also:** "About User-Managed Media Recovery" on page 17-14 for more information about applying redo log files

Performing Datafile Recovery in an Open Database

It is possible for a media failure to occur while the database remains open, leaving the undamaged datafiles online and available for use. Damaged datafiles—but not

the tablespaces that contain them—are automatically taken offlineif the database writer is unable to write to them. Queries that cannot read damaged files return errors, but the datafiles are not taken offline because of the failed queries. For example, you may run a query and see output such as:

```
ERROR at line 1:
ORA-01116: error in opening database file 3
ORA-01110: data file 11: '/oracle/oradata/trgt/cwmlite02.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
```

The procedure in this section cannot be used to perform complete media recovery on the datafiles of the SYSTEM tablespace while the database is open. If the media failure damages datafiles of the SYSTEM tablespace, then the database automatically shuts down.

Perform media recovery in these stages:

1. Prepare the database for recovery by making sure it is open and taking the tablespaces requiring recovery offline, as described in "Preparing for Open Database Recovery" on page 17-25.
2. Restore the necessary files in the affected tablespaces as described in "Restoring Backups of the Inaccessible Datafiles" on page 17-26.
3. Recover the affected tablespaces as described in "Recovering Offline Tablespaces in an Open Database" on page 17-26.

See Also:

- "Determining Which Datafiles Require Recovery" on page 17-4
- "Performing Closed Database Recovery" on page 17-21 for procedures for proceeding with complete media recovery of the SYSTEM tablespace

Preparing for Open Database Recovery

In this stage, you take affected tablespaces offline and inspect the media device that is causing the problem.

To prepare for datafile recovery when the database is open:

1. If the database is open when you discover that recovery is required, take all tablespaces containing damaged datafiles offline. For example, if tablespace `users` and `tools` contain damaged datafiles, enter:

```
ALTER TABLESPACE users OFFLINE TEMPORARY;
ALTER TABLESPACE tools OFFLINE TEMPORARY;
```

2. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, proceed with database recovery by restoring damaged files to an alternative storage device.

Restoring Backups of the Inaccessible Datafiles

In this stage, you restore all necessary backups in the offline tablespaces.

To restore datafiles in an open database:

1. If files are permanently damaged, then restore the most recent backup files of *only* the datafiles damaged by the media failure. Do not restore undamaged datafiles, online redo logs, or control files. If the hardware problem is fixed and the datafiles can be restored to their original locations, then do so. Otherwise, restore the datafiles to an alternative storage device.

> **Note:** In some circumstances, if you do not have a backup of a specific datafile, you can use ALTER DATABASE CREATE DATAFILE to create an empty replacement file that is recoverable.

2. If you restored one or more damaged datafiles to alternative locations, update the control file of the database to reflect the new datafile names. For example, to change the filename of the datafile in tablespace `users` you might enter:

```
ALTER DATABASE RENAME FILE '?/oradata/trgt/users01.dbf' TO
                           '/disk2/users01.dbf';
```

> **See Also:** *Oracle Database SQL Reference* for more information about ALTER DATABASE RENAME FILE

Recovering Offline Tablespaces in an Open Database

In the final stage, you recover the datafiles in the offline tablespaces.

To recover offline tablespaces in an open database:

1. Connect to the database with administrator privileges, and start offline tablespace recovery of all damaged datafiles in one or more offline tablespaces using one step. For example, recover `users` and `tools`:

   ```
   RECOVER TABLESPACE users, tools  # recovers datafiles in users and tools
   ```

   Note: For best performance, use parallel recovery to recover the datafiles. See "Performing Media Recovery in Parallel" on page 17-39.

2. The database begins the roll forward phase of media recovery by applying the necessary redo logs (archived and online) to reconstruct the restored datafiles. Unless the applying of files is automated with `RECOVER AUTOMATIC` or `SET AUTORECOVERY ON`, the database prompts for each required redo log file.

   Recovery continues until all required archived logs have been applied to the datafiles. The online redo logs are then automatically applied to the restored datafiles to complete media recovery. If no archived redo logs are required for complete media recovery, then the database does not prompt for any. Instead, all necessary online redo logs are applied, and media recovery is complete.

3. When the damaged tablespaces are recovered up to the moment that media failure occurred, bring the offline tablespaces online. For example, to bring tablespaces `users` and `tools` online, issue the following statements:

   ```
   ALTER TABLESPACE users ONLINE;
   ALTER TABLESPACE tools ONLINE;
   ```

   See Also: *Oracle Database Administrator's Guide* for more information about creating datafiles

Performing Incomplete User-Managed Media Recovery

This section describes the steps necessary to complete the different types of incomplete media recovery operations, and includes the following topics:

- Preparing for Incomplete Recovery
- Restoring Datafiles Before Performing Incomplete Recovery
- Performing Cancel-Based Incomplete Recovery
- Performing Time-Based or Change-Based Incomplete Recovery

Note: If your database is affected by seasonal time changes (for example, daylight savings time), then you may experience a problem if a time appears twice in the redo log and you want to recover to the second, or later time. To handle time changes, perform cancel-based or change-based recovery.

Preparing for Incomplete Recovery

In this phase, you examine the source of the media problem.

To prepare for incomplete recovery:

1. If you are uncertain about performing incomplete recovery, then back up the whole database—all datafiles, a control file, and the parameter files—as a precautionary measure in case an error occurs during the recovery procedure.
2. If the database is still open and incomplete media recovery is necessary, then terminate the instance:

   ```
   SHUTDOWN ABORT
   ```

3. If a media failure occurred, correct the hardware problem that caused the failure. If the hardware problem cannot be repaired quickly, then proceed with database recovery by restoring damaged files to an alternative storage device.

Restoring Datafiles Before Performing Incomplete Recovery

In this phase, you restore a whole database backup.

To restore the files necessary for cancel-based recovery and bring them online:

1. If the current control files do not match the physical structure of the database at the intended time of recovery, then restore a backup control file as described in "Losing All Current Control Files When a Backup Is Available" on page 17-9. The restored control file should reflect the database's physical file structure at the point at which incomplete media recovery should finish. To determine which control file backup to use:
   - Review the list of files that corresponds to the current control file and each control file backup to determine the correct control file to use.
   - If necessary, replace all current control files of the database with the correct control file backup.
   - Alternatively, create a new control file to replace the missing one.

> **Note:** If you are unable to restore a control file backup to one of the CONTROL_FILES locations, then edit the initialization parameter file so that this CONTROL_FILES location is removed.

2. Restore backups of *all* the datafiles of the database. All backups used to replace existing datafiles must have been taken before the intended time of recovery. For example, if you intend to recover to January 2 at 2:00 p.m., then restore all datafiles with backups completed before this time. Follow these guidelines:

If . . .	Then . . .
You do not have a backup of a datafile	Create an empty replacement file that can be recovered as described in "Restoring Backups of the Damaged or Missing Files" on page 17-22.
A datafile was added after the intended time of recovery	Do not restore a backup of this file because it will no longer be used for the database after recovery completes.
The hardware problem causing the failure has been solved and all datafiles can be restored to their default locations	Restore the files as described in "Restoring Datafiles and Archived Redo Logs" on page 17-5 and skip Step 4 of this procedure.
A hardware problem persists	Restore damaged datafiles to an alternative storage device.

> **Note:** Files in read-only tablespaces should be offline if you are using a control file backup. Otherwise, the recovery will try to update the headers of the read-only files.

3. Start SQL*Plus and connect to the database with administrator privilege, then start a new instance and mount the database:

```
STARTUP MOUNT
```

4. If one or more damaged datafiles were restored to alternative locations in Step 2, then indicate the new locations of these files to the control file of the associated database. For example, enter:

```
ALTER DATABASE RENAME FILE '?/oradata/trgt/users01.dbf' TO
                           '/disk2/users01.dbf';
```

5. Obtain the datafile names and statuses of all datafiles by checking the list of datafiles that normally accompanies the current control file or querying the V$DATAFILE view. For example, enter:

```
SELECT NAME,STATUS FROM V$DATAFILE;
```

6. Ensure that all datafiles of the database are online. All datafiles of the database requiring recovery must be online unless a tablespace was taken offline with the NORMAL option or is a read-only tablespace. For example, to guarantee that a datafile named `?/oradata/trgt/users01.dbf` is online, enter the following:

```
ALTER DATABASE DATAFILE '?/oradata/trgt/users01.dbf' ONLINE;
```

   If a specified datafile is already online, the statement has no effect. If you prefer, create a script to bring all datafiles online at once as in the following:

```
SPOOL onlineall.sql
SELECT 'ALTER DATABASE DATAFILE '''||name||''' ONLINE;' FROM V$DATAFILE;
SPOOL OFF
SQL> @onlineall
```

Performing Cancel-Based Incomplete Recovery

In cancel-based recovery, recovery proceeds by prompting you with the suggested filenames of archived redo log files. Recovery stops when you specify CANCEL instead of a filename or when all redo has been applied to the datafiles.

Cancel-based recovery is better than change-based or time-based recovery if you want to control which archived log terminates recovery. For example, you may know that you have lost all logs past sequence 1234, so you want to cancel recovery after log 1233 is applied.

You should perform cancel-based media recovery in these stages:

1. Prepare for recovery by backing up the database and correct any media failures as described in "Preparing for Incomplete Recovery" on page 17-28.
2. Restore backup datafiles as described in "Restoring Datafiles Before Performing Incomplete Recovery" on page 17-28. If you have a current control file, then do not restore a backup control file.
3. Perform media recovery on the restored database backup as described in the following procedure.

To perform cancel-based recovery:

1. Start SQL*Plus and connect to the database with administrator privileges, then start a new instance and mount the database:

   ```
   STARTUP MOUNT
   ```

2. Begin cancel-based recovery by issuing the following command:

   ```
   RECOVER DATABASE UNTIL CANCEL
   ```

   If you are using a backup control file with this incomplete recovery, then specify the USING BACKUP CONTROLFILE option in the RECOVER command.

   ```
   RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE
   ```

   > **Note:** If you fail to specify the UNTIL clause on the RECOVER command, then the database assumes a complete recovery and will not open until all redo is applied.

3. The database applies the necessary redo log files to reconstruct the restored datafiles. The database supplies the name it expects to find from LOG_ARCHIVE_DEST_1 and requests you to stop or proceed with applying the log file. Note that if the control file is a backup, then you must supply the names of the online logs if you want to apply the changes in these logs.

   > **Note:** If you use a Real Application Clusters (RAC) configuration, and you are performing incomplete recovery or using a backup control file, then the database can only compute the name of the first archived redo log file from the *first* thread. The first redo log file from the other threads must be supplied by the user. After the first log file in a given thread has been supplied, the database can suggest the names of the subsequent log files in this thread.

4. Continue applying redo log files until the last log has been applied to the restored datafiles, then cancel recovery by executing the following command:

   ```
   CANCEL
   ```

   The database indicates whether recovery is successful. If you cancel before all the datafiles have been recovered to a consistent SCN and then try to open the database, you will get an ORA-1113 error if more recovery is necessary. As

explained in "Determining Which Datafiles Require Recovery" on page 17-4, you can query V$RECOVER_FILE to determine whether more recovery is needed, or if a backup of a datafile was not restored prior to starting incomplete recovery.

5. Open the database with the RESETLOGS option. You must always reset the logs after incomplete recovery or recovery with a backup control file. For example:

```
ALTER DATABASE OPEN RESETLOGS;
```

See Also: "Opening the Database with the RESETLOGS Option" on page 17-33

Performing Time-Based or Change-Based Incomplete Recovery

This section describes how to perform the time-based media recovery procedure in the following stages:

1. Prepare for recovery by backing up the database and correct any media failures as described in "Preparing for Incomplete Recovery" on page 17-28.
2. Restore backup datafiles as described in "Restoring Datafiles Before Performing Incomplete Recovery" on page 17-28. If you have a current control file, then do not restore a backup control file.
3. Perform media recovery with the following procedure.

To perform change-based or time-based recovery:

1. Issue the RECOVER DATABASE UNTIL statement to begin recovery. If recovering to an SCN, specify as a decimal number without quotation marks. For example, to recover through SCN 10034 issue:

```
RECOVER DATABASE UNTIL CHANGE 10034;
```

If recovering to a time, the time is always specified using the following format, delimited by single quotation marks: 'YYYY-MM-DD:HH24:MI:SS'. The following statement recovers the database up to a specified time:

```
RECOVER DATABASE UNTIL TIME '2000-12-31:12:47:30'
```

2. Apply the necessary redo log files to recover the restored datafiles. The database automatically terminates the recovery when it reaches the correct time, and returns a message indicating whether recovery is successful.

Note: Unless recovery is automated, the database supplies the name from LOG_ARCHIVE_DEST_1 and asks you to stop or proceed with after each log. If the control file is a backup, then after the archived logs are applied you must supply the names of the online logs.

3. Open the database in RESETLOGS mode. You must always reset the online logs after incomplete recovery or recovery with a backup control file. For example:

```
ALTER DATABASE OPEN RESETLOGS;
```

See Also: "Opening the Database with the RESETLOGS Option" on page 17-33

Opening the Database with the RESETLOGS Option

Whenever you perform incomplete recovery or recovery with a backup control file, you must reset the online logs when you open the database. The new version of the reset database is called a new **incarnation**.

This section contains the following topics:

About Opening with the RESETLOGS Option

The RESETLOGS option is always *required* after incomplete media recovery or recovery using a backup control file. Resetting the redo log does the following:

- Archives the current online redo logs (if they are accessible) and then erases the contents of the online redo logs and resets the log sequence number to 1. For example, if the current online redo logs are sequence 1000 and 1001 when you open RESETLOGS, then the database archives logs 1000 and 1001 and then resets the online logs to sequence 1 and 2.
- Creates the online redo log files if they do not currently exist.
- Reinitializes the control file metadata about online redo logs and redo threads.

- Updates all current datafiles and online redo logs and all subsequent archived redo logs with a new RESETLOGS SCN and time stamp.

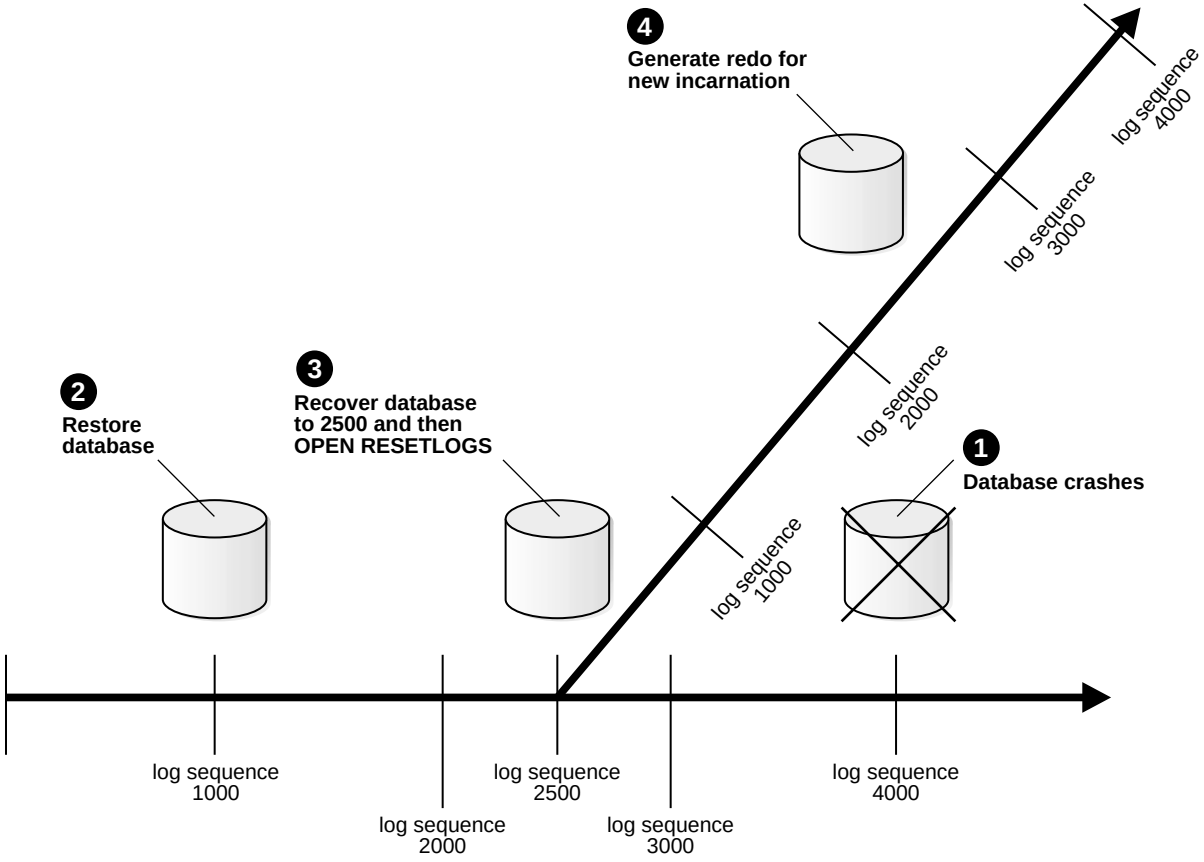
Because the database will not apply an archived log to a datafile unless the RESETLOGS SCN and time stamps match, the RESETLOGS prevents you from corrupting datafiles with archived logs that are not from direct parent incarnations of the current incarnation.

In prior releases, it was recommended that you back up the database immediately after the RESETLOGS. Because you can now easily recover a pre-RESETLOGS backup like any other backup, making a new database backup is optional. In order to perform recovery through resetlogs you must have all archived logs generated since the last backup and at least one control file (current, backup, or created).

Figure 17–1 shows the case of a database that can only be recovered to log sequence 2500 because an archived redo log is missing. When the online redo log is at sequence 4000, the database crashes. You restore the sequence 1000 backup and prepare for complete recovery. Unfortunately, one of your archived logs is corrupted. The log before the missing log contains sequence 2500, so you recover to this log sequence and open RESETLOGS. As part of the RESETLOGS, the database archives the current online logs (sequence 4000 and 4001) and resets the log sequence to 1.

You generate changes in the new incarnation of the database, eventually reaching log sequence 4000. The changes between sequence 2500 and sequence 4000 for the new incarnation of the database are different from the changes between sequence 2500 and sequence 4000 for the old incarnation. You cannot apply logs generated *after* 2500 in the old incarnation to the new incarnation, but you can apply the logs generated *before* 2500 in the old incarnation to the new incarnation. The invalid logs are said to be **orphaned** in the new incarnation because they are unusable for recovery.

Figure 17–1 Creating a New Database Incarnation



Executing the ALTER DATABASE OPEN Statements

To preserve the log sequence number when opening a database after media recovery, execute either of the following statements:

```
ALTER DATABASE OPEN NORESETLOGS;
ALTER DATABASE OPEN;
```

To reset the log sequence number when opening a database after recovery and thereby create a new incarnation of the database, execute the following statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

If you open with the RESETLOGS option, the database returns different messages depending on whether recovery was complete or incomplete. If the recovery was complete, then the following message appears in the `alert_SID.log` file:

```
RESETLOGS after complete recovery through change scn
```

If the recovery was incomplete, then this message is reported in the `alert_SID.log` file, where *scn* refers to the end point of incomplete recovery:

```
RESETLOGS after incomplete recovery UNTIL CHANGE scn
```

If you attempt to OPEN RESETLOGS when you should not, or if you neglect to reset the log when you should, then the database returns an error and does not open the database. Correct the problem and try again.

> **See Also:** "About User-Managed Media Recovery Problems" on page 20-2 for descriptions of situations that can cause ALTER DATABASE OPEN RESETLOGS to fail

Checking the Alert Log After a RESETLOGS Operation

After opening the database with the RESETLOGS option, check the `alert_SID.log` to see whether the database detected inconsistencies between the data dictionary and the control file, for example, a datafile that the data dictionary includes but which is not listed in the new control file. The following table describes two possible scenarios.

Control File	Data Dictionary	Result
Datafile is listed	Datafile is not listed	References to the unlisted datafile are removed from the control file. A message in the alert log indicates what was found.
Datafile is not listed	Datafile is listed	The database creates a placeholder entry in the control file under `MISSING`*nnnnn* (where *nnnnn* is the file number in decimal). `MISSING`*nnnnn* is flagged in the control file as offline and requiring media recovery. You can make the datafile corresponding to `MISSING`*nnnnn* accessible by using ALTER DATABASE RENAME FILE for `MISSING`*nnnnn* so that it points to the datafile. If you do not have a backup of this datafile, then drop the tablespace.

Recovering a Database in NOARCHIVELOG Mode

If a media failure damages datafiles in a NOARCHIVELOG database, then the only option for recovery is usually to restore a consistent whole database backup. If you are using logical backups created by an Oracle export utility to supplement regular physical backups, then you can also attempt to restore the database by importing an exported backup of the database into a re-created database or a database restored from an old backup.

Restoring a NOARCHIVELOG Database to its Default Location

In this scenario, the media failure is repaired so that you are able to restore all database files to their original location.

To restore the most recent whole database backup to the default location:

1. If the database is open, then shut down the database. For example, enter:

   ```
   SHUTDOWN IMMEDIATE
   ```

2. If possible, correct the media problem so that the backup database files can be restored to their original locations.

3. Restore the most recent whole database backup with operating system commands as described in "Restoring Datafiles and Archived Redo Logs" on page 17-5. Restore all of the datafiles and control files of the whole database backup, not just the damaged files. The following example restores a whole database backup to its default location:

   ```
   % cp /backup/*.dbf $ORACLE_HOME/oradata/trgt/
   ```

4. Because online redo logs are not backed up, you cannot restore them with the datafiles and control files. In order to allow the database to reset the online redo logs, you must first mimic incomplete recovery:

   ```
   RECOVER DATABASE UNTIL CANCEL
   CANCEL
   ```

5. Open the database in RESETLOGS mode:

   ```
   ALTER DATABASE OPEN RESETLOGS;
   ```

Restoring a NOARCHIVELOG Database to a New Location

In this scenario, you restore the database files to an alternative location because the original location is damaged by a media failure.

To restore the most recent whole database backup to a new location:

1. If the database is open, then shut it down. For example, enter:

   ```
   SHUTDOWN IMMEDIATE
   ```

2. Restore all of the datafiles and control files of the whole database backup, not just the damaged files. If the hardware problem has not been corrected and some or all of the database files must be restored to alternative locations, then restore the whole database backup to a new location. For example, enter:

   ```
   % cp /backup/*.dbf /new_disk/oradata/trgt/
   ```

3. If necessary, edit the restored parameter file to indicate the new location of the control files. For example:

   ```
   CONTROL_FILES = "/new_disk/oradata/trgt/control01.dbf"
   ```

4. Start an instance using the restored and edited parameter file and mount, but do not open, the database. For example:

   ```
   STARTUP MOUNT
   ```

5. If the restored datafile filenames will be different (as will be the case when you restore to a different file system or directory, on the same node or a different node), then update the control file to reflect the new datafile locations. For example, to rename datafile 1 you might enter:

   ```
   ALTER DATABASE RENAME FILE '?/oradata/trgt/system01.dbf' TO
                              '/new_disk/oradata/system01.dbf';
   ```

6. If the online redo logs were located on a damaged disk, and the hardware problem is not corrected, then specify a new location for each affected online log. For example, enter:

   ```
   ALTER DATABASE RENAME FILE '?/oradata/trgt/redo01.log' TO
                              '/new_disk/oradata/redo_01.log';
   ALTER DATABASE RENAME FILE '?/oradata/trgt/redo02.log' TO
                              '/new_disk/oradata/redo_02.log';
   ```

7. Because online redo logs are not backed up, you cannot restore them with the datafiles and control files. In order to allow the database to reset the online redo logs, you must first mimic incomplete recovery:

   ```
   RECOVER DATABASE UNTIL CANCEL;
   CANCEL;
   ```

8. Open the database in RESETLOGS mode. This command clears the online redo logs and resets the log sequence to 1:

```
ALTER DATABASE OPEN RESETLOGS;
```

Note that restoring a NOARCHIVELOG database backup and then resetting the log discards all changes to the database made from the time the backup was taken to the time of the failure.

> **See Also:** *Oracle Database Administrator's Guide* for more information about renaming and relocating datafiles, and *Oracle Database SQL Reference* to learn about ALTER DATABASE RENAME FILE

Performing Media Recovery in Parallel

Use **parallel media recovery** to tune the roll forward phase of media recovery. In parallel media recovery, the database uses a "division of labor" approach to allocate different processes to different data blocks while rolling forward, thereby making the procedure more efficient. For example, if parallel recovery is performed with PARALLEL 4, and only one datafile is recovered, then four spawned processes read blocks from the datafile and apply records instead of only one process.

Typically, recovery is I/O-bound on reads to data blocks. Parallelism at the block level may only help recovery performance if it increases total I/Os, for example, by bypassing operating system restrictions on asynchronous I/Os. Systems with efficient asynchronous I/O see little benefit from parallel media recovery.

The SQL*Plus RECOVER PARALLEL command specifies parallel media recovery (the default is NOPARALLEL). This command selects a degree of parallelism equal to the number of CPUs available on all participating instances times the value of the PARALLEL_THREADS_PER_CPU initialization parameter.

The format for the RECOVER PARALLEL command is the following:

```
RECOVER PARALLEL integer;
```

The *integer* variable sets the number of recovery processes used for media recovery. If you use a Real Application Clusters configuration, then the database decides how to distribute these recovery processes among the instances. If *integer* is not specified, then the database picks a default number of recovery processes.

Note: The RECOVERY_PARALLELISM initialization parameter specifies the number of concurrent recovery processes for instance or crash recovery *only*. Media recovery is not affected.

See Also:

- *Oracle Database Performance Tuning Guide* for more information on parallel recovery
- *SQL*Plus User's Guide and Reference* for more information about the SQL*Plus RECOVER . . . PARALLEL statement

18

Advanced User-Managed Recovery Scenarios

This chapter describes how to recover from common media failures, and includes the following topics:

Recovering After the Loss of Datafiles: Scenarios

If a media failure affects datafiles, then the recovery procedure depends on:

- The archiving mode of the database: ARCHIVELOG or NOARCHIVELOG
- The type of media failure
- The files affected by the media failure

Losing Datafiles in NOARCHIVELOG Mode

If either a permanent or temporary media failure affects any datafiles of a database operating in NOARCHIVELOG mode, then the database automatically shuts down.

If the media failure is temporary, correct the underlying problem and restart the database. Usually, crash recovery will recover all committed transactions from the online redo log. If the media failure is permanent, then restore the database as described in "Recovering a Database in NOARCHIVELOG Mode" on page 17-37.

Losing Datafiles in ARCHIVELOG Mode

If either a permanent or temporary media failure affects the datafiles of a database operating in ARCHIVELOG mode, then the following scenarios can occur.

Damaged Datafiles	Database Status	Solution
Datafiles in the SYSTEM tablespace or datafiles with active undo segments.	Database shuts down.	If the hardware problem is temporary, then fix it and restart the database. Usually, crash recovery recovers lost transactions. If the hardware problem is permanent, then recover the database as described in "Performing Closed Database Recovery" on page 17-21.
Datafiles not in the SYSTEM tablespace or datafiles that do not contain active rollback or undo segments.	Affected datafiles are taken offline, but the database stays open.	If the unaffected portions of the database must remain available, then do not shut down the database. Take tablespaces containing problem datafiles offline using the temporary option, then recover them as described in "Performing Datafile Recovery in an Open Database" on page 17-24.

Recovering Through an Added Datafile with a Backup Control File: Scenario

If database recovery with a backup control file rolls forward through a CREATE TABLESPACE or an ALTER TABLESPACE ADD DATAFILE operation, then the database stops recovery when applying the redo record for the added files and lets you confirm the filenames.

For example, suppose the following sequence of events occurs:

1. You back up the database
2. You create a new tablespace containing two datafiles: `/oracle/oradata/trgt/test01.dbf` and `/oracle/oradata/trgt/test02.dbf`.
3. You later restore a backup control file and perform media recovery through the CREATE TABLESPACE operation.

You may see the following error when applying the CREATE TABLESPACE redo data:

```
ORA-00283: recovery session canceled due to errors
ORA-01244: unnamed datafile(s) added to controlfile by media recovery
ORA-01110: data file 11: '/oracle/oradata/trgt/test02.dbf'
ORA-01110: data file 10: '/oracle/oradata/trgt/test01.dbf'
```

To recover through an ADD DATAFILE operation:

1. View the files added by selecting from V$DATAFILE. For example:

```
SELECT FILE#,NAME
FROM V$DATAFILE;

FILE#           NAME
--------------- ----------------------
1               /oracle/oradata/trgt/system01.dbf
.
.
.
10               /oracle/oradata/trgt/UNNAMED00001
11               /oracle/oradata/trgt/UNNAMED00002
```

2. If multiple unnamed files exist, then determine which unnamed file corresponds to which datafile by using one of these methods:

- Open the `alert_SID.log`, which contains messages about the original file location for each unnamed file.
- Derive the original file location of each unnamed file from the error message and V$DATAFILE: each unnamed file corresponds to the file in the error message with the same file number.

3. Issue the ALTER DATABASE RENAME FILE statement to rename the datafiles. For example, enter:

```
ALTER DATABASE RENAME FILE '/db/UNNAMED00001' TO
                           '/oracle/oradata/trgt/test01.dbf';
ALTER DATABASE RENAME FILE '/db/UNNAMED00002' TO
                           '/oracle/oradata/trgt/test02.dbf';
```

4. Continue recovery by issuing the previous recovery statement. For example:

```
RECOVER AUTOMATIC DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL
```

Re-Creating Datafiles When Backups Are Unavailable: Scenario

If a datafile is damaged and no backup of the file is available, then you can still recover the datafile if:

- All archived log files written after the creation of the original datafile are available
- The control file contains the name of the damaged file (that is, the control file is current, or is a backup taken after the damaged datafile was added to the database)

> **Note:** You cannot re-create any of the datafiles for the SYSTEM tablespace by using the CREATE DATAFILE clause of the ALTER DATABASE statement because the necessary redo is not available.

To re-create a datafile for recovery:

1. Create a new, empty datafile to replace a damaged datafile that has no corresponding backup. For example, assume that the datafile `?/oradata/trgt/users01.dbf` has been damaged, and no backup is available. The following statement re-creates the original datafile (same size) on `disk2`:

```
ALTER DATABASE CREATE DATAFILE '?/oradata/trgt/users01.dbf' AS
```

```
'/disk2/users01.dbf';
```

This statement creates an empty file that is the same size as the lost file. The database looks at information in the control file and the data dictionary to obtain size information. The old datafile is renamed as the new datafile.

2. Perform media recovery on the empty datafile. For example, enter:

```
RECOVER DATAFILE '/disk2/users01.dbf'
```

3. All archived logs written after the original datafile was created must be applied to the new, empty version of the lost datafile during recovery.

Recovering Through RESETLOGS with Created Control File: Scenario

You can recover backups through an OPEN RESETLOGS so long as:

- You have a current, backup, or created control file that knows about the prior incarnations
- You have all available archived redo logs

If you need to re-create the control file, the trace file generated by ALTER DATABASE BACKUP CONTROLFILE TO TRACE will contain the necessary commands to re-construct the complete incarnation history. The V$DATABASE_INCARNATION view displays the RESETLOGS history known to the control file, while the V$LOG_HISTORY view displays the archived log history.

It is possible for the incarnation history to be incomplete in the in re-created control file. For example, archived logs necessary for recovery may be missing. In this case, it is possible to create incarnation records explicitly with the ALTER DATABASE REGISTER LOGFILE statement.

In the following example, you register four logs that are necessary for recovery but are not recorded in the re-created control file, and then recover the database:

```
ALTER DATABASE REGISTER LOGFILE '?/oradata/trgt/arch/arcr_1_1_42343523.arc';
ALTER DATABASE REGISTER LOGFILE '?/oradata/trgt/arch/arcr_1_1_34546466.arc';
ALTER DATABASE REGISTER LOGFILE '?/oradata/trgt/arch/arcr_1_1_23435466.arc';
ALTER DATABASE REGISTER LOGFILE '?/oradata/trgt/arch/arcr_1_1_12343533.arc';
RECOVER AUTOMATIC DATABASE;
```

Recovering NOLOGGING Tables and Indexes: Scenario

You can create tables and indexes with the CREATE TABLE AS SELECT statement. You can also specify that the database create them with the NOLOGGING option. When you create a table or index as NOLOGGING, the database does not generate redo log records for the operation. Thus, you cannot recover objects created with NOLOGGING, even if you are running in ARCHIVELOG mode.

Note: If you cannot afford to lose tables or indexes created with NOLOGGING, then make a backup after the unrecoverable table or index is created.

Be aware that when you perform media recovery, and some tables or indexes are created normally whereas others are created with the NOLOGGING option, the NOLOGGING objects are marked logically corrupt by the RECOVER operation. Any attempt to access the unrecoverable objects returns an ORA-01578 error message. Drop the NOLOGGING objects and re-create them if needed.

Because it is possible to create a table with the NOLOGGING option and then create an index with the LOGGING option on that table, the index is not marked as logically corrupt after you perform media recovery. The table was unrecoverable (and thus marked as corrupt after recovery), however, so the index points to corrupt blocks. The index must be dropped, and the table and index must be re-created if necessary.

See Also: *Oracle Data Guard Concepts and Administration* for information about the impact of NOLOGGING on a b database

Recovering Read-Only Tablespaces with a Backup Control File: Scenario

If you have a read-only tablespace on read-only or slow media, then you may encounter errors or poor performance when recovering with the USING BACKUP CONTROLFILE option. This situation occurs when the backup control file indicates that a tablespace was read / write when the control file was backed up. In this case, media recovery may attempt to write to the files. For read-only media, the database issues an error saying that it cannot write to the files. For slow media, such as a hierarchical storage system backed up by tapes, performance may suffer.

To avoid these recovery problems, use current control files rather than backups to recover the database. If you need to use a backup control file, then you can also avoid this problem if the read-only tablespace has not suffered a media failure.

Recovery of Read-Only or Slow Media with a Backup Control File

You have these alternatives for recovering read-only and slow media when using a backup control file:

- Take datafiles from read-only tablespaces offline before doing recovery with a backup control file, and then bring the files online at the end of media recovery.
- Use the correct version of the control file for the recovery. If the tablespace will be read-only when recovery completes, then the control file backup must be from a time when the tablespace was read-only. Similarly, if the tablespace will be read / write at the end of recovery, then the control file must be from a time when the tablespace was read / write.

Recovery of Read-Only Files with a Re-Created Control File

If a current or backup control file is unavailable for recovery, then you can execute a `CREATE CONTROLFILE` statement as described in "Losing All Current and Backup Control Files" on page 17-12. Read-only files should not be listed in the `CREATE CONTROLFILE` statement so that recovery can skip these files. No recovery is required for read-only datafiles unless you restored backups of these files from a time when the datafiles were read / write.

After you create a new control file and attempt to mount and open the database, the database performs a data dictionary check against the files listed in the control file. Files that were not listed in the `CREATE CONTROLFILE` statement but are present in the data dictionary have entries created for them in the control file. These files are named as `MISSING`*nnnnn*, where *nnnnn* is a five digit number starting with `0`.

After the database is open, rename the read-only files to their correct filenames by executing the `ALTER DATABASE RENAME FILE` statement for all the files whose name is prefixed with `MISSING`.

To prepare for a scenario in which you might have to re-create the control file, run the following statement when the database is mounted or open to obtain the `CREATE CONTROLFILE` syntax:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

This SQL statement produces a trace file that you can edit and use as a script to re-create the control file. You can specify either the RESETLOGS or NORESETLOGS (default) keywords to generate CREATE CONTROLFILE ... RESETLOGS or CREATE CONTROLFILE ... NORESETLOGS versions of the script.

All the restrictions related to read-only files in CREATE CONTROLFILE statements also apply to offline normal tablespaces, except that you need to bring the tablespace online after the database is open. You should leave out tempfiles from the CREATE CONTROLFILE statement and add them after database open.

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how to make trace backups of the control file

Recovering Transportable Tablespaces: Scenario

The **transportable tablespace** feature of Oracle allows a user to transport a set of tablespaces from one database to another. Transporting a tablespace into a database is like creating a tablespace with preloaded data. Using this feature is often an advantage because:

- It is faster than using the Export or SQL*Loader utilities because it involves only copying datafiles and integrating metadata
- You can use it to move index data, hence avoiding the necessity of rebuilding indexes

Like normal tablespaces, transportable tablespaces are recoverable. While you can recover normal tablespaces without a backup, you must have a version of the transported datafiles in order to recover a transported tablespace.

To recover a transportable tablespace:

1. If the database is open, then take the transported tablespace offline. For example, if you want to recover the users tablespace, then issue:

   ```
   ALTER TABLESPACE users OFFLINE IMMEDIATE;
   ```

2. Restore a backup of the transported datafiles with an operating system utility. The backup can be the initial version of the transported datafiles or any backup taken after the tablespace is transported. For example, enter:

   ```
   % cp /backup/users.dbf $ORACLE_HOME/oradata/trgt/users01.dbf
   ```

3. Recover the tablespace as normal. For example, enter:

   ```
   RECOVER TABLESPACE users
   ```

You may see the error ORA-01244 when recovering through a transportable tablespace operation just as when recovering through a CREATE TABLESPACE operation. In this case, rename the unnamed files to the correct locations using the procedure in "Recovering Through an Added Datafile with a Backup Control File: Scenario" on page 18-3.

> **See Also:** *Oracle Database Administrator's Guide* for detailed information about using the transportable tablespace feature

Recovering After the Loss of Online Redo Log Files: Scenarios

If a media failure has affected the online redo logs of a database, then the appropriate recovery procedure depends on the following:

- The configuration of the online redo log: mirrored or non-mirrored
- The type of media failure: temporary or permanent
- The types of online redo log files affected by the media failure: current, active, unarchived, or inactive

Table 18–1 displays V$LOG status information that can be crucial in a recovery situation involving online redo logs.

Table 18–1 STATUS Column of V$LOG

Status	Description
UNUSED	The online redo log has never been written to.
CURRENT	The online redo log is active, that is, needed for instance recovery, and it is the log to which the database is currently writing. The redo log can be open or closed.
ACTIVE	The online redo log is active, that is, needed for instance recovery, but is not the log to which the database is currently writing.It may be in use for block recovery, and may or may not be archived.
CLEARING	The log is being re-created as an empty log after an ALTER DATABASE CLEAR LOGFILE statement. After the log is cleared, then the status changes to UNUSED.
CLEARING_CURRENT	The current log is being cleared of a closed thread. The log can stay in this status if there is some failure in the switch such as an I/O error writing the new log header.
INACTIVE	The log is no longer needed for instance recovery. It may be in use for media recovery, and may or may not be archived.

Recovering After Losing a Member of a Multiplexed Online Redo Log Group

If the online redo log of a database is multiplexed, and if at least one member of each online redo log group is not affected by the media failure, then the database continues functioning as normal, but error messages are written to the log writer trace file and the `alert_SID.log` of the database.

Solve the problem by taking one of the following actions:

- If the hardware problem is temporary, then correct it. The log writer process accesses the previously unavailable online redo log files as if the problem never existed.
- If the hardware problem is permanent, then drop the damaged member and add a new member by using the following procedure.

> **Note:** The newly added member provides no redundancy until the log group is reused.

To replace a damaged member of a redo log group:

1. Locate the filename of the damaged member in `V$LOGFILE`. The status is `INVALID` if the file is inaccessible:

```
SELECT GROUP#, STATUS, MEMBER
FROM V$LOGFILE
WHERE STATUS='INVALID';

GROUP#    STATUS       MEMBER
-------   -----------  --------------------
0002      INVALID      /oracle/oradata/trgt/redo02.log
```

2. Drop the damaged member. For example, to drop member `redo01.log` from group 2, issue:

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/oradata/trgt/redo02.log';
```

3. Add a new member to the group. For example, to add `redo02.log` to group 2, issue:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/oradata/trgt/redo02b.log'
  TO GROUP 2;
```

   If the file you want to add already exists, then it must be the same size as the other group members, and you must specify `REUSE`. For example:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/oradata/trgt/redo02b.log'
  REUSE TO GROUP 2;
```

Recovering After the Loss of All Members of an Online Redo Log Group

If a media failure damages all members of an online redo log group, then different scenarios can occur depending on the type of online redo log group affected by the failure and the archiving mode of the database.

If the damaged log group is inactive, then it is not needed for crash recovery; if it is active, then it is needed for crash recovery.

If the group is . . .	Then . . .	And you should . . .
Inactive	It is not needed for crash recovery	Clear the archived or unarchived group.
Active	It is needed for crash recovery	Attempt to issue a checkpoint and clear the log; if impossible, then you must restore a backup and perform incomplete recovery up to the most recent available redo log.
Current	It is the log that the database is currently writing to	Attempt to clear the log; if impossible, then you must restore a backup and perform incomplete recovery up to the most recent available redo log.

Your first task is to determine whether the damaged group is active or inactive.

To determine whether the damaged groups are active:

1. Locate the filename of the lost redo log in V$LOGFILE and then look for the group number corresponding to it. For example, enter:

```
SELECT GROUP#, STATUS, MEMBER FROM V$LOGFILE;

GROUP#    STATUS       MEMBER
-------   -----------  ---------------------
0001                    /oracle/dbs/log1a.f
0001                    /oracle/dbs/log1b.f
0002      INVALID       /oracle/dbs/log2a.f
0002      INVALID       /oracle/dbs/log2b.f
0003                    /oracle/dbs/log3a.f
0003                    /oracle/dbs/log3b.f
```

2. Determine which groups are active. For example, enter:

```
SELECT GROUP#, MEMBERS, STATUS, ARCHIVED
FROM V$LOG;

GROUP#  MEMBERS          STATUS     ARCHIVED
------  -------          ---------  -----------
 0001   2                INACTIVE   YES
 0002   2                ACTIVE     NO
 0003   2                CURRENT    NO
```

3. If the affected group is inactive, follow the procedure in "Losing an Inactive Online Redo Log Group" on page 18-12. If the affected group is active (as in the preceding example), then follow the procedure in "Losing an Active Online Redo Log Group" on page 18-14.

Losing an Inactive Online Redo Log Group

If all members of an online redo log group with INACTIVE status are damaged, then the procedure depends on whether you can fix the media problem that damaged the inactive redo log group.

If the failure is . . .	Then . . .
Temporary	Fix the problem. LGWR can reuse the redo log group when required.
Permanent	The damaged inactive online redo log group eventually halts normal database operation. Reinitialize the damaged group manually by issuing the ALTER DATABASE CLEAR LOGFILE statement as described in this section.

Clearing Inactive, Archived Redo You can clear an inactive redo log group when the database is open or closed. The procedure depends on whether the damaged group has been archived.

To clear an inactive, online redo log group that has been archived:

1. If the database is shut down, then start a new instance and mount the database:

   ```
   STARTUP MOUNT
   ```

2. Reinitialize the damaged log group. For example, to clear redo log group 2, issue the following statement:

   ```
   ALTER DATABASE CLEAR LOGFILE GROUP 2;
   ```

Clearing Inactive, Not-Yet-Archived Redo Clearing a not-yet-archived redo log allows it to be reused without archiving it. This action makes backups unusable if they were started before the last change in the log, unless the file was taken offline prior to the first change in the log. Hence, if you need the cleared log file for recovery of a backup, then you cannot recover that backup. Also, it prevents complete recovery from backups due to the missing log.

To clear an inactive, online redo log group that has not been archived:

1. If the database is shut down, then start a new instance and mount the database:

   ```
   STARTUP MOUNT
   ```

2. Clear the log using the UNARCHIVED keyword. For example, to clear log group 2, issue:

   ```
   ALTER DATABASE CLEAR LOGFILE UNARCHIVED GROUP 2;
   ```

   If there is an offline datafile that requires the cleared log to bring it online, then the keywords UNRECOVERABLE DATAFILE are required. The datafile and its entire tablespace have to be dropped because the redo necessary to bring it online is being cleared, and there is no copy of it. For example, enter:

   ```
   ALTER DATABASE CLEAR LOGFILE UNARCHIVED GROUP 2 UNRECOVERABLE DATAFILE;
   ```

3. Immediately back up the whole database with an operating system utility, so that you have a backup you can use for complete recovery without relying on the cleared log group. For example, enter:

   ```
   % cp /disk1/oracle/dbs/*.f /disk2/backup
   ```

4. Back up the database's control file with the ALTER DATABASE statement. For example, enter:

   ```
   ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/dbs/cf_backup.f';
   ```

Failure of CLEAR LOGFILE Operation The ALTER DATABASE CLEAR LOGFILE statement can fail with an I/O error due to media failure when it is not possible to:

- Relocate the redo log file onto alternative media by re-creating it under the currently configured redo log filename
- Reuse the currently configured log filename to re-create the redo log file because the name itself is invalid or unusable (for example, due to media failure)

In these cases, the ALTER DATABASE CLEAR LOGFILE statement (before receiving the I/O error) would have successfully informed the control file that the log was being cleared and did not require archiving. The I/O error occurred at the step in which the CLEAR LOGFILE statement attempts to create the new redo log file and write zeros to it. This fact is reflected in V$LOG.CLEARING_CURRENT.

Losing an Active Online Redo Log Group

If the database is still running and the lost active redo log is *not* the current log, then issue the ALTER SYSTEM CHECKPOINT statement. If successful, then the active redo log is rendered inactive, and you can follow the procedure in "Losing an Inactive Online Redo Log Group" on page 18-12. If unsuccessful, or if your database has halted, then perform one of procedures in this section, depending on the archiving mode.

The current log is the one LGWR is currently writing to. If a LGWR I/O fails, then LGWR terminates and the instance crashes. In this case, you must restore a backup, perform incomplete recovery, and open the database with the RESETLOGS option.

To recover from loss of an active online log group in NOARCHIVELOG mode:

1. If the media failure is temporary, then correct the problem so that the database can reuse the group when required.
2. Restore the database from a consistent, whole database backup (datafiles and control files) as described in "Restoring Datafiles Before Performing Incomplete Recovery" on page 17-28. For example, enter:

   ```
   % cp /disk2/backup/*.dbf $ORACLE_HOME/oradata/trgt/
   ```

3. Mount the database:

   ```
   STARTUP MOUNT
   ```

4. Because online redo logs are not backed up, you cannot restore them with the datafiles and control files. In order to allow the database to reset the online redo logs, you must first mimic incomplete recovery:

   ```
   RECOVER DATABASE UNTIL CANCEL
   CANCEL
   ```

5. Open the database using the RESETLOGS option:

   ```
   ALTER DATABASE OPEN RESETLOGS;
   ```

6. Shut down the database consistently. For example, enter:

```
SHUTDOWN IMMEDIATE
```

7. Make a whole database backup.

To recover from loss of an active online redo log group in ARCHIVELOG mode:

If the media failure is temporary, then correct the problem so that the database can reuse the group when required. If the media failure is not temporary, then use the following procedure.

1. Begin incomplete media recovery, recovering up through the log before the damaged log.
2. Ensure that the current name of the lost redo log can be used for a newly created file. If not, then rename the members of the damaged online redo log group to a new location. For example, enter:

```
ALTER DATABASE RENAME FILE "?/oradata/trgt/redo01.log" TO "/tmp/redo01.log";
ALTER DATABASE RENAME FILE "?/oradata/trgt/redo01.log" TO "/tmp/redo02.log";
```

3. Open the database using the RESETLOGS option:

```
ALTER DATABASE OPEN RESETLOGS;
```

> **Note:** All updates executed from the endpoint of the incomplete recovery to the present must be re-executed.

Loss of Multiple Redo Log Groups

If you have lost multiple groups of the online redo log, then use the recovery method for the most difficult log to recover. The order of difficulty, from most difficult to least difficult, follows:

1. The current online redo log
2. An active online redo log
3. An unarchived online redo log
4. An inactive online redo log

Recovering After the Loss of Archived Redo Log Files: Scenario

If the database is operating in ARCHIVELOG mode, and if the only copy of an archived redo log file is damaged, then the damaged file does not affect the present

operation of the database. The following situations can arise, however, depending on when the redo log was written and when you backed up the datafile.

If you backed up . . .	Then . . .
All datafiles after the filled online redo log group (which is now archived) was written	The archived version of the filled online redo log group is not required for complete media recovery operation.
A specific datafile before the filled online redo log group was written	If the corresponding datafile is damaged by a permanent media failure, use the most recent backup of the damaged datafile and perform incomplete recovery of the tablespace containing the damaged datafile, up to the damaged log.

Caution: If you know that an archived redo log group has been damaged, immediately back up all datafiles so that you will have a whole database backup that does not require the damaged archived redo log.

Recovering from a Dropped Table: Scenario

One not-uncommon error is the accidental dropping of a table from your database. In general, the fastest and simplest solution is to use the flashback drop feature, described in "Oracle Flashback Drop: Undo a DROP TABLE Operation" on page 9-6, to reverse the dropping of the table. However, if for some reason, such as flashback drop being disabled or the table having been dropped with the PURGE option, you cannot use flashback table, you can create a copy of the database, perform point-in-time recovery of that copy to a time before the table was dropped, export the dropped table using an Oracle export utility, and re-import it into your primary database using an Oralce import utility.

In this scenario, assume that you do not have the flashback database functionality enabled, so FLASHBACK DATABASE is not an option, but you do have physical backups of the database.

Note: If you have granted powerful privileges (such as DROP ANY TABLE) to only selected, appropriate users, you can minimize user errors that require database recovery.

To recover a table that has been accidentally dropped:

1. If possible, keep the database that experienced the user error online and available for use. Back up all datafiles of the existing database in case an error is made during the remaining steps of this procedure.
2. Restore a database backup to an alternative location, then perform incomplete recovery of this backup using a restored backup control file, to the point just before the table was dropped.
3. Export the lost data from the temporary, restored version of the database using an Oracle export utility. In this case, export the accidentally dropped table.

   Note: System audit options are exported.

4. Use an Oracle import utility to import the data back into the production database.
5. Delete the files of the temporary copy of the database to conserve space.

See Also: *Oracle Database Utilities* for more information about the Oracle export and import utilities

Performing Media Recovery in a Distributed Environment: Scenario

How you perform media recovery depends on whether your database participates in a distributed database system. The Oracle distributed database architecture is autonomous. Therefore, depending on the type of recovery operation selected for a single damaged database, you may have to coordinate recovery operations globally among all databases in the distributed system.

Table 18–2 summarizes different types of recovery operations and whether coordination among nodes of a distributed database system is required.

Table 18–2 Recovery Operations in a Distributed Database Environment

If you are . . .	Then . . .
Restoring a whole backup for a database that was never accessed from a remote node	Use non-coordinated, autonomous database recovery.
Restoring a whole backup for a database that was accessed by a remote node for a database in `NOARCHIVELOG` mode	Shut down all databases and restore them using the same coordinated full backup.
Performing complete media recovery of one or more databases in a distributed database	Use non-coordinated, autonomous database recovery.
Performing incomplete media recovery of a database that was never accessed by a remote node	Use non-coordinated, autonomous database recovery.
Performing incomplete media recovery of a database that was accessed by a remote node	Use coordinated, incomplete recovery to the same global point in time for all databases in the distributed system.

Coordinating Time-Based and Change-Based Distributed Database Recovery

If one node in a distributed database requires recovery to a past time, it is often necessary to recover all other nodes in the system to the same point in time to preserve global data consistency. This operation is called **coordinated, time-based, distributed database recovery**. The following tasks should be performed with the standard procedures of time-based and change-based recovery described in this chapter.

1. Recover the database that requires the recovery operation using time-based recovery. For example, if a database needs to be recovered because of a media failure, then recover this database first using time-based recovery. Do not recover the other databases at this point.
2. After you have recovered the database and opened it with the `RESETLOGS` option, search the `alert_`*`SID`*`.log` of the database for the `RESETLOGS` message.

   If the message is, "`RESETLOGS after complete recovery through change xxx`", then you have applied all the changes in the database and performed complete recovery. Do not recover any of the other databases in the distributed system, or you will unnecessarily remove changes in them. Recovery is complete.

   If the message is, "`RESETLOGS after incomplete recovery UNTIL CHANGE xxx`", then you have successfully performed an incomplete recovery. Record the change number from the message and proceed to the next step.

3. Recover all other databases in the distributed database system using change-based recovery, specifying the change number (SCN) from Step 2.

Dropping a Database with SQL*Plus

You may need to remove a database, that is, the database files that form the database, from the operating system. For example, this scenario can occur when you create a test database and then no longer have a use for it. The SQL*Plus command DROP DATABASE can perform this function.

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how to use the equivalent RMAN command DROP DATABASE

To drop the database:

1. Start SQL*Plus and connect to the target database with administrator privileges, then ensure that the database is either mounted or open with no users connected. For example:

   ```
   SQL> STARTUP FORCE MOUNT
   ```

2. Remove the datafiles and control files listed in the control file from the operating system. For example:

   ```
   SQL> DROP DATABASE; # deletes all database files, both ASM and non-ASM
   ```

   If the database is on raw disk, the command does not delete the actual raw disk special files.

3. Use an operating system utility to delete all backups and archived logs associated with the database because these are not automatically deleted by the SQL*Plus command. For example:

   ```
   % rm /backup/* ?/oradata/trgt/arch/*
   ```

19

Performing User-Managed TSPITR

This chapter describes how to perform user-managed tablespace point-in-time recovery (TSPITR) with the transportable tablespace feature.

This chapter includes the following topics:

- Introduction to User-Managed Tablespace Point-in-Time Recovery
- Preparing for Tablespace Point-in-Time Recovery: Basic Steps
- Restoring and Recovering the Auxiliary Database: Basic Steps
- Performing TSPITR with Transportable Tablespaces
- Performing Partial TSPITR of Partitioned Tables
- Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped
- Performing TSPITR of Partitioned Tables When a Partition Has Split

> **See Also:** Chapter 10, "RMAN Tablespace Point-in-Time Recovery (TSPITR)"

Introduction to User-Managed Tablespace Point-in-Time Recovery

Tablespace point-in-time recovery (TSPITR) with the transportable tablespace feature enables you to quickly recover one or more tablespaces (other than the SYSTEM tablespace) to a time that is prior to the rest of the database.

User-managed TSPITR is most useful for recovering the following:

- An erroneous DROP TABLESPACE operation
- An incorrect batch job or other DML statement that has affected only a subset of the database
- A logical schema to a point different from the rest of the physical database when multiple schemas exist in separate tablespaces of one physical database
- A tablespace in a VLDB (very large database) when TSPITR is more efficient than restoring the whole database from a backup and rolling it forward

Refer to "Preparing for Tablespace Point-in-Time Recovery: Basic Steps" on page 19-4 before deciding to perform TSPITR.

TSPITR Terminology

Familiarize yourself with the following terms and abbreviations, which are used throughout this chapter:

TSPITR
Tablespace point-in-time recovery

Primary Database
The database containing the tablespace or tablespaces that you want to recover to a prior point in time.

Auxiliary Database
A copy of the current database that is restored from a backup. It includes restored backups on the auxiliary host of the following files:

- Datafiles belonging to the SYSTEM tablespace
- Datafiles in the set of tablespaces to be recovered
- Datafiles belonging to an undo tablespace or tablespace that contains rollback segments

All backups must be from a point in time prior to the desired recovery time.

Recovery Set
All the tablespaces on the primary database that require point-in-time recovery to be performed on them.

Recovery Set Self-Containment Check
All objects that are part of the recovery set must be self-contained: there can be no dependencies on objects outside the recovery set. For example, if a table is part of the recovery set and its indexes are in a separate tablespace, then the recovery set must include the tablespace containing the index. Alternatively, the index can be dropped. You can check the recovery set tablespaces for self-containment with the procedure `DBMS_TTS.TRANSPORT_SET_CHECK`.

Auxiliary Set
Any other files required for restoring the auxiliary database, including:

- Backup control file
- Datafiles from the `SYSTEM` tablespace
- Datafiles in an undo tablespace or datafiles containing rollback segments

Transportable Tablespace
A rapid method of transporting tablespaces across databases by unplugging them from a source database and plugging them into a target database. The databases can even be on different platforms, for example, Solaris and Windows 2000. The unplugging and plugging is done with the Export and Import utilities. Note that there is no actual export and import of the table data, but simply an export and import of internal metadata. During the procedure, the datafiles of the transported tablespaces are made part of the target database.

TSPITR Methods

In releases prior to Oracle9*i*, you had the following two methods for performing user-managed TSPITR:

- Traditional user-managed TSPITR, which required you to create a special type of database called a **clone database**
- User-managed TSPITR with the transportable tablespace feature

As of Oracle Database Release 10*g*, TSPITR should be performed by using the transportable tablespace feature. This procedure is relatively easy to use and is less error prone than the traditional method, which is currently deprecated (although not yet unsupported).

TSPITR is performed by dropping the tablespaces to be recovered from the primary database, restoring a copy of the database called an **auxiliary database** and recovering it to the desired point in time, then transporting the relevant tablespaces from the auxiliary database to the current version of the primary database.

For ease of use, it is highly recommended that you place the auxiliary and primary databases on different hosts. Nevertheless, you can also perform TSPITR when the databases are located on the same host.

The basic procedure for performing user-managed TSPITR is as follows:

1. Take the tablespaces requiring TSPITR offline.
2. Plan the setup of the auxiliary database.
3. Create the auxiliary database and recover it to the desired point in time.
4. Drop the tablespaces requiring TSPITR from the primary database.
5. Use the transportable tablespace feature to transport the set of tablespaces from the auxiliary database to the primary database.

> **See Also:** *Oracle Database Administrator's Guide* for a complete account of how to use the transportable tablespace feature

Preparing for Tablespace Point-in-Time Recovery: Basic Steps

TSPITR requires careful planning. Before proceeding you should read this chapter thoroughly.

This section contains the following topics:

Caution: You should not perform TSPITR for the first time on a production system, or when there is a time constraint.

Step 1: Review TSPITR Requirements

Satisfy the following requirements before performing TSPITR:

- Ensure that you have backups of all datafiles in the recovery and auxiliary set tablespaces. The datafile backups must have been created *before* the desired TSPITR time.
- Ensure that you have a control file backup that is usable on the auxiliary database. To be usable, the control file must meet these requirements:
  - The control file must have been backed up *before* the desired TSPITR time.
  - The control file must have been backed up with the following SQL statement, where *cf_name* refers to the fully specified filename:

    ```
    ALTER DATABASE BACKUP CONTROLFILE TO 'cf_name';
    ```

- Ensure that all files constituting the recovery set tablespaces are in the recovery set on the auxiliary database; otherwise, the export phase during tablespace transport fails.
- Allocate enough disk space on the auxiliary host to accommodate the auxiliary database.
- Provide enough real memory to start the auxiliary instance.
- If the tablespace to be recovered has been renamed, ensure that the target SCN for TSPITR is after the time when the file was renamed. You cannot TSPITR a renamed tablespace to a point in time earlier than the rename. However, you can perform a DBPITR to an SCN before the rename. In this case, the tablespace reverts to its name as of the target SCN.

See Also: "Step 6: Create the Initialization Parameter File for the Auxiliary Instance" on page 19-7

Step 2: Identify All of the Files in the Recovery and Auxiliary Set Tablespaces

Before you create the auxiliary database, make sure that you connect to the primary database with administrator privileges and obtain all of the following information about the primary database:

- The filenames of the datafiles in the recovery set tablespaces
- The filenames of the datafiles in the SYSTEM tablespace
- The filenames of the datafiles in an undo tablespace or datafiles containing rollback segments

- The filenames of the control files

The following useful query displays the filenames of all datafiles and control files in the database:

```
SELECT NAME FROM V$DATAFILE
UNION ALL
SELECT NAME FROM V$CONTROLFILE;
```

To determine the filenames of the datafiles in the SYSTEM and recovery set tablespaces, execute the following query and replace *RECO_TBS_1*, *RECO_TBS_2*, and so forth with the names of the recovery set tablespaces:

```
SELECT t.NAME AS "reco_tbs", d.NAME AS "dbf_name"
       FROM V$DATAFILE d, V$TABLESPACE t
WHERE t.TS# = d.TS#
AND t.NAME IN ('SYSTEM', 'RECO_TBS_1', 'RECO_TBS_2');
```

If you run the database in manual undo management mode (which is deprecated), then the following query displays the names of the tablespaces containing rollback segments as well as the names of the datafiles in the tablespaces:

```
SELECT DISTINCT r.TABLESPACE_NAME AS "rbs_tbs", d.FILE_NAME AS "dbf_name"
       FROM DBA_ROLLBACK_SEGS r, DBA_DATA_FILES d
WHERE r.TABLESPACE_NAME=d.TABLESPACE_NAME;
```

If you run the database in automatic undo management mode, then the following query displays the names of the undo tablespaces as well as the names of the datafiles in the tablespaces:

```
SELECT DISTINCT u.TABLESPACE_NAME AS "undo_tbs", d.FILE_NAME AS "dbf_name"
       FROM DBA_UNDO_EXTENTS u, DBA_DATA_FILES d
WHERE u.TABLESPACE_NAME=d.TABLESPACE_NAME;
```

Step 3: Determine Whether Objects Will Be Lost

When TSPITR is performed on a tablespace, any objects created after the recovery time are lost. To determine which objects will be lost, query the TS_PITR_OBJECTS_TO_BE_DROPPED view on the primary database. The contents of the view are described in Table 19–1.

Table 19–1 TS_PITR_OBJECTS_TO_BE_DROPPED View

Column Name	Meaning
OWNER	Owner of the object to be dropped.

Table 19–1 TS_PITR_OBJECTS_TO_BE_DROPPED View (Cont.)

Column Name	Meaning
NAME	The name of the object that will be lost as a result of TSPITR
CREATION_TIME	Creation time stamp for the object.
TABLESPACE_NAME	Name of the tablespace containing the object.

When querying this view, supply all the elements of the date field, otherwise the default setting is used. Also, use the TO_CHAR and TO_DATE functions. For example, with a recovery set consisting of users and tools, and a recovery point in time of 19 October 2002, 15:34:11, execute the following SQL script:

```
SELECT OWNER, NAME, TABLESPACE_NAME,
       TO_CHAR(CREATION_TIME, 'YYYY-MM-DD:HH24:MI:SS')
       FROM SYS.TS_PITR_OBJECTS_TO_BE_DROPPED
WHERE TABLESPACE_NAME IN ('users','tools')
AND CREATION_TIME > TO_DATE('02-OCT-19:15:34:11','YY-MON-DD:HH24:MI:SS')
ORDER BY TABLESPACE_NAME, CREATION_TIME;
```

See Also: *Oracle Database Reference* for more information about the TS_PITR_OBJECTS_TO_BE_DROPPED view

Step 4: Choose a Method for Connecting to the Auxiliary Instance

You must be able to connect to the auxiliary instance. You can either use Oracle Net or operating system authentication. To learn how to configure networking files, refer to *Oracle Net Services Administrator's Guide*.

Step 5: Create an Oracle Password File for the Auxiliary Instance

For information about creating and maintaining Oracle password files, refer to the *Oracle Database Administrator's Guide*. If you do not use a password file, then you can skip this step.

Step 6: Create the Initialization Parameter File for the Auxiliary Instance

Create a new initialization parameter file rather than copying and then editing the production database initialization parameter file. Save memory by using low settings for parameters such as the following:

- DB_CACHE_SIZE

- SHARED_POOL_SIZE
- LARGE_POOL_SIZE

Reducing the preceding parameter settings can prevent the auxiliary database from starting when other dependent parameters are set too high—for example, the initialization parameter ENQUEUE_RESOURCES, which allocates memory from within the shared pool.

The auxiliary database can be either on the same host as the primary database or on a different host. Because the auxiliary database filenames are identical to the primary database filenames in the auxiliary control file, you must update the auxiliary control file to point to the locations to which the files were restored for the auxiliary database. If the auxiliary database is on the same machine as the primary database, or if the auxiliary database is on a different machine that uses different path names, then you must rename the control files, datafiles, and online redo logs. If the auxiliary database is on a different machine with the same path names, then you can rename just the online redo logs. To view the names of the online redo log files of the primary database so that you can be sure to use unique names when creating the auxiliary, use this query on the primary database:

```
SELECT NAME FROM V$LOGFILE;
```

> **Caution:** If the auxiliary and primary database are on the same machine, then failing to rename the online redo log files may cause primary database corruption.

Set the parameters shown in Table 19–2 in the auxiliary initialization parameter file.

Table 19–2 Auxiliary Initialization Parameters

Parameter	Purpose
DB_NAME	Names the auxiliary database. Leave the name of the auxiliary database *the same as* the primary database.
CONTROL_FILES	Identifies auxiliary control files. Set to the filename of the auxiliary control file. If the auxiliary database is on the same host as the primary database, make sure that the control file name is *different from* the primary database control file name.
DB_UNIQUE_NAME	Allows the auxiliary database to start even though it has the same name as the primary database. Set to any unique value, for example, = AUX. This parameter is only needed if the auxiliary and primary database are on the same host.

Table 19–2 Auxiliary Initialization Parameters (Cont.)

Parameter	Purpose
DB_FILE_NAME_CONVERT	Uses patterns to convert filenames for the datafiles of the auxiliary database. This parameter is only necessary if you are either restoring the auxiliary database on the same host as the primary host, or on a different host that uses different path names from the primary host.
LOG_FILE_NAME_CONVERT	Uses patterns to convert filenames for the online redo logs of the auxiliary database. This parameter is mandatory.
LOG_ARCHIVE_DEST_1	Specifies the default directory containing the archived redo logs required for recovery.This parameter specifies the location on the auxiliary host in which the archived logs will be located.
LOG_ARCHIVE_FORMAT	Specifies the format of the archived logs. You should use the same format setting used in the primary initialization parameter file.

Set other parameters as needed, including the parameters that allow you to connect as SYSDBA through Oracle Net.

For example, the auxiliary parameter file for a database on the same host as the primary could look like the following:

```
DB_NAME = prod1
CONTROL_FILES = /oracle/aux/control01.dbf
DB_UNIQUE_NAME = aux
DB_FILE_NAME_CONVERT=("/oracle/oradata/","/aux/")
LOG_FILE_NAME_CONVERT=("/oracle/oradata/","/aux/")
LOG_ARCHIVE_DEST_1 = 'LOCATION=/oracle/oradata/arch/'
LOG_ARCHIVE_FORMAT = arcr_%t_%s.arc
```

The auxiliary parameter file for a database on a different host with the same path names as the primary could look like the following:

```
DB_NAME = prod1
# you do not need to set CONTROL_FILES or DB_FILE_NAME_CONVERT because the file
# system structure on both hosts is identical
LOG_FILE_NAME_CONVERT=("/oracle/oradata/","/tmp/oradata/")
LOG_ARCHIVE_DEST_1 = 'LOCATION=/tmp/arch/'
LOG_ARCHIVE_FORMAT = arcr_%t_%s.arc
```

Restoring and Recovering the Auxiliary Database: Basic Steps

The procedure for restore and recovery of the auxiliary database differs depending on whether the auxiliary database is on the same host as the primary database. The examples in this section assume:

- You are performing TSPITR on production database called `prod1` located on host `prim_host`.
- The recovery set tablespaces are `users` and `tools`. Tablespace `users` contains datafile `/oracle/oradata/users01.dbf` and tablespace `tools` contains datafile `/fs2/tools01.dbf`.
- The auxiliary set contains the `SYSTEM` tablespace datafile `/oracle/oradata/system.dbf`, the undo tablespace datafile `/oracle/oradata/undo01.dbf`, and the control file `/oracle/oradata/control01.dbf`.
- The online redo logs are named `/oracle/oradata/redo01.log` and `/oracle/oradata/redo02.log`.
- All the primary database files are contained in `/oracle/oradata`

The different cases are described in the following sections:

- Restoring and Recovering the Auxiliary Database on the Same Host
- Restoring the Auxiliary Database on a Different Host with the Same Path Names
- Restoring the Auxiliary Database on a Different Host with Different Path Names

Restoring and Recovering the Auxiliary Database on the Same Host

The following examples assume the case in which you restore the auxiliary database to the same host as the primary database. In this scenario, all of the primary database files are contained in `/oracle/oradata`, and you want to restore the auxiliary database to `/oracle/oradata/aux`. So, you set `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` to convert the filenames from `/oracle/oradata` to `/oracle/oradata/aux`.

Perform the following tasks to restore and recover the auxiliary database:

1. Restore the auxiliary set and the recovery set to a location different from that of the primary database. For example, assume that the auxiliary set consists of the following files:

```
/oracle/oradata/control01.dbf    # control file
/oracle/oradata/undo01.dbf       # datafile in undo tablespace
/oracle/oradata/system.dbf       # datafile in SYSTEM tablespace
```

And the recovery set consists of the following datafiles:

```
/oracle/oradata/users01.dbf  # datafile in users tablespace
/oracle/oradata/tools01.dbf  # datafile in tools tablespace
```

You can restore backups of the auxiliary set files and recovery set files to a new location as follows:

```
cp /backup/control01.dbf /oracle/oradata/aux/control01.dbf
cp /backup/undo01.dbf /oracle/oradata/aux/undo01.dbf
cp /backup/system.dbf /oracle/oradata/aux/system.dbf
cp /backup/users01.dbf /oracle/oradata/aux/users01.dbf
cp /backup/tools01.dbf /oracle/oradata/aux/tools01.dbf
```

2. Start the auxiliary database without mounting it, specifying the initialization parameter file if necessary. For example, enter:

   ```
   STARTUP NOMOUNT PFILE=/aux/initAUX.ora
   ```

3. Mount the auxiliary database, specifying the CLONE keyword:

   ```
   ALTER DATABASE MOUNT CLONE DATABASE;
   ```

   The CLONE keyword causes Oracle to take all datafiles offline automatically.

4. Manually rename all auxiliary database files to reflect their new locations *only if* these files are not renamed by DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT. In our scenario, all datafiles and online redo logs are renamed by initialization parameters, so no manual renaming is necessary.

5. Run the following SQL script on the auxiliary database to ensure that all datafiles are named correctly:

   ```
   SELECT NAME FROM V$DATAFILE
   UNION ALL
   SELECT MEMBER FROM V$LOGFILE
   UNION ALL
   SELECT NAME FROM V$CONTROLFILE
   /
   ```

   If not, then rename the files manually as in the previous step.

6. Bring only the datafiles in the auxiliary and recovery set tablespaces online. For example, bring the four datafiles in the recovery and auxiliary sets online:

```
ALTER DATABASE DATAFILE /oracle/oradata/aux/system.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/aux/users01.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/aux/tools01.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/aux/undo01.dbf ONLINE;
```

> **Note:** The export phase of TSPITR will not work if all the files of each recovery set tablespace are not online.

   At this point, the auxiliary database is mounted and ready for media recovery.

7. Recover the auxiliary database to the specified point in time with the USING BACKUP CONTROLFILE option. Use any form of incomplete recovery. The following example uses cancel-based incomplete recovery:

```
RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE
```

8. Open the auxiliary database with the RESETLOGS option using the following statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

Restoring the Auxiliary Database on a Different Host with the Same Path Names

The following example assumes that you create the auxiliary database on a different host called aux_host. The auxiliary host has the same path names as the primary host. Hence, you do not need to rename the auxiliary database datafiles. So, you do *not* need to set DB_FILE_NAME_CONVERT, although you should set LOG_FILE_NAME_CONVERT.

To restore and recover the auxiliary database:

1. Restore the auxiliary set and the recovery set to the auxiliary host. For example, assume that the auxiliary set consists of the following files:

```
/oracle/oradata/control01.dbf      # control file
/oracle/oradata/undo01.dbf     # datafile in undo tablespace
/oracle/oradata/system.dbf   # datafile in SYSTEM tablespace
```

   And the recovery set consists of the following datafiles:

```
/oracle/oradata/users01.dbf  # 1st datafile in users tablespace
/oracle/oradata/tools01.dbf  # 2nd datafile in tools tablespace
```

These files will occupy the same locations in the auxiliary host.

2. Start the auxiliary database without mounting it, specifying the initialization parameter file if necessary. For example, enter:

```
STARTUP NOMOUNT PFILE=/aux/initAUX.ora
```

3. Mount the auxiliary database, specifying the CLONE keyword:

```
ALTER DATABASE MOUNT CLONE DATABASE;
```

The CLONE keyword causes Oracle to take all datafiles offline automatically.

4. Rename all auxiliary database files to reflect their new locations *only if* these files are not renamed by DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT. In our scenario, the datafiles do not require renaming, and the logs are converted with LOG_FILE_NAME_CONVERT. So, no manual renaming is necessary.

5. Run the following script in SQL*Plus on the auxiliary database to ensure that all datafiles are named correctly.

```
SELECT NAME FROM V$DATAFILE
UNION ALL
SELECT MEMBER FROM V$LOGFILE
UNION ALL
SELECT NAME FROM V$CONTROLFILE
;
```

If not, then rename them manually as in the previous step.

6. Bring all datafiles in the auxiliary and recovery set tablespaces online. For example, bring the four datafiles in the recovery and auxiliary sets online:

```
ALTER DATABASE DATAFILE /oracle/oradata/system.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/users01.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/tools01.dbf ONLINE;
ALTER DATABASE DATAFILE /oracle/oradata/undo01.dbf ONLINE;
```

Note: The export phase of TSPITR will not work if all the files of each recovery set tablespace are not online.

At this point, the auxiliary database is mounted and ready for media recovery.

7. Recover the auxiliary database to the specified point in time with the USING BACKUP CONTROLFILE option. Use any form of incomplete recovery. The following example uses cancel-based incomplete recovery:

```
RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE
```

8. Open the auxiliary database with the RESETLOGS option using the following statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

Restoring the Auxiliary Database on a Different Host with Different Path Names

This case should be treated exactly like "Restoring and Recovering the Auxiliary Database on the Same Host" on page 19-10. The same guidelines for renaming files apply in both cases.

Performing TSPITR with Transportable Tablespaces

After you have completed the preparation stage, begin the actual TSPITR procedure as described in *Oracle Database Administrator's Guide*. The procedure occurs in the following steps:

- Step 1: Unplugging the Tablespaces from the Auxiliary Database
- Step 2: Transporting the Tablespaces into the Primary Database

Step 1: Unplugging the Tablespaces from the Auxiliary Database

In this step, you recover the auxiliary database to the desired past time, then unplug the desired tablespaces.

To unplug the auxiliary database tablespaces:

1. Connect SQL*Plus to the auxiliary database with administrator privileges. For example:

```
% sqlplus 'SYS/oracle@aux AS SYSDBA'
```

2. Make the tablespaces in the recovery set read-only by running the ALTER TABLESPACE . . . READ ONLY statement. For example, make users and tools read-only as follows:

```
ALTER TABLESPACE users READ ONLY;
ALTER TABLESPACE tools READ ONLY;
```

3. Ensure that the recovery set is self-contained. For example:

```
EXECUTE SYS.DBMS_TTS.TRANSPORT_SET_CHECK('users,tools',TRUE,TRUE);
```

4. Query the transportable tablespace violations table to manage any dependencies. For example:

```
SELECT * FROM SYS.TRANSPORT_SET_VIOLATIONS;
```

   This query should return no rows after all dependencies are managed. Refer to *Oracle Database Administrator's Guide* for more information about this table.

5. Generate the transportable set by running the Export utility as described in *Oracle Database Administrator's Guide*. Include all tablespaces in the recovery set, as in the following example:

```
% exp SYS/oracle TRANSPORT_TABLESPACE=y TABLESPACES=(users,tools) \
TTS_FULL_CHECK=y
```

   This command generates an export file named `expdat.dmp`.

Step 2: Transporting the Tablespaces into the Primary Database

In this step, you transport the recovery set tablespaces into the primary database.

To plug the recovery set tablespaces into the primary database:

1. Connect SQL*Plus to the primary database (*not* the auxiliary database). For example:

```
% sqlplus 'SYS/oracle@primary AS SYSDBA'
```

2. Drop the tablespaces in the recovery set with the DROP TABLESPACE statement. For example:

```
DROP TABLESPACE users INCLUDING CONTENTS;
DROP TABLESPACE tools INCLUDING CONTENTS;
```

3. Restore the recovery set datafiles from the auxiliary database to the recovery set file locations in the primary database. For example:

```
% cp /net/aux_host/aux/users01.dbf \
> /net/primary_host/oracle/oradata/users01.dbf
% cp /net/aux_host/aux/tools01.dbf \
> /net/primary_host/oracle/oradata/tools01.dbf
```

4. Move the export file `expdat.dmp` to the primary host. For example, enter:

```
% cp /net/aux_host/aux/expdat.dmp \
> /net/primary_host/oracle/oradata/expdat.dmp
```

5. Plug in the transportable set into the primary database by running Import as described in *Oracle Database Administrator's Guide*. For example:

```
% imp SYS/oracle TRANSPORT_TABLESPACE=y FILE=expat.dmp
   DATAFILES=('/oracle/oradata/users01.dbf','/oracle/oradata/tools01.dbf')
```

6. Make the recovered tablespaces read / write by executing the `ALTER TABLESPACE READ WRITE` statement. For example:

```
ALTER TABLESPACE users READ WRITE;
ALTER TABLESPACE tools READ WRITE;
```

7. Back up the recovered tablespaces with an operating system utility.

Caution: You must back up the tablespace because otherwise you might lose it. For example, a media failure occurs, but the archived logs from the last backup of the database do not logically link to the recovered tablespaces. If you attempt to recover any recovery set tablespaces from a backup taken before TSPITR, then recovery fails.

Performing Partial TSPITR of Partitioned Tables

Partitioned tables can span multiple tablespaces. Follow this procedure only if the recovery set does not fully contain all of the partitions.

This section describes how to perform partial TSPITR of partitioned tables that have a range that has not changed or expanded, and includes the following steps:

- Step 1: Create a Table on the Primary Database for Each Partition Being Recovered
- Step 2: Drop the Indexes on the Partition Being Recovered
- Step 3: Exchange Partitions with Standalone Tables
- Step 4: Drop the Recovery Set Tablespace
- Step 5: Create Tables at Auxiliary Database
- Step 6: Drop Indexes on Partitions Being Recovered

- Step 7: Exchange Partitions with Standalone Tables on the Auxiliary Database
- Step 8: Transport the Recovery Set Tablespaces
- Step 9: Exchange Partitions with Standalone Tables on the Primary Database
- Step 10: Back Up the Recovered Tablespaces in the Primary Database

> **Note:** Often you have to recover the dropped partition along with recovering a partition whose range has expanded. Refer to "Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped" on page 19-19.

Step 1: Create a Table on the Primary Database for Each Partition Being Recovered

This table should have the exact same column names and column datatypes as the partitioned table you are recovering. Create the table using the following template:

```
CREATE TABLE new_table AS
  SELECT * FROM partitioned_table
  WHERE 1=2;
```

These tables are used to swap each recovery set partition (see "Step 3: Exchange Partitions with Standalone Tables" on page 19-17).

> **Note:** The table and the partition must belong to the same schema.

Step 2: Drop the Indexes on the Partition Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, nonpartitioned indexes that exist on the partition you wish to recover. If you drop the indexes on the partition being recovered, then you need to drop them on the auxiliary database (see "Step 6: Drop Indexes on Partitions Being Recovered" on page 19-18). Rebuild the indexes after TSPITR is complete.

Step 3: Exchange Partitions with Standalone Tables

Exchange each partition in the recovery set with its associated standalone table (created in "Step 1: Create a Table on the Primary Database for Each Partition Being Recovered" on page 19-17) by issuing the following statement, replacing the variables with the names of the appropriate objects:

```
ALTER TABLE table_name EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

Step 4: Drop the Recovery Set Tablespace

On the primary database, drop each tablespace in the recovery set. For example, enter the following, replacing *tablespace_name* with the name of the tablespace:

```
DROP TABLESPACE tablespace_name INCLUDING CONTENTS;
```

Step 5: Create Tables at Auxiliary Database

After recovering the auxiliary database and opening it with the RESETLOGS option, create a table in the SYSTEM tablespace that has the same column names and column data types as the partitioned table you are recovering. You must create the table in the SYSTEM tablespace: otherwise, Oracle issues the ORA-01552 error.

Create a table for each partition you wish to recover. These tables are used later to swap each recovery set partition.

Note: The table and the partition must belong to the same schema.

Step 6: Drop Indexes on Partitions Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, non-partitioned indexes that exist on the partition you wish to recover (on the table created in "Step 1: Create a Table on the Primary Database for Each Partition Being Recovered" on page 19-17).

Step 7: Exchange Partitions with Standalone Tables on the Auxiliary Database

For each partition in the auxiliary database recovery set, exchange the partitions with the standalone tables (created in "Step 5: Create Tables at Auxiliary Database" on page 19-18) by executing the following SQL script, replacing the variables with the appropriate object names:

```
ALTER TABLE partitioned_table_name
EXCHANGE PARTITION partition_name
WITH TABLE table_name;
```

Step 8: Transport the Recovery Set Tablespaces

Export the recovery set tablespaces from the auxiliary database and then import them into the primary database as described in "Performing TSPITR with Transportable Tablespaces" on page 19-14.

Step 9: Exchange Partitions with Standalone Tables on the Primary Database

For each recovered partition on the primary database, swap its associated standalone table with the following statement, replacing the variables with the appropriate object names:

```
ALTER TABLE table_name EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

If the associated indexes have been dropped, then re-create them.

Step 10: Back Up the Recovered Tablespaces in the Primary Database

Back up the recovered tablespaces on the primary database. Failure to do so results in loss of data in the event of media failure.

Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped

This section describes how to perform TSPITR on partitioned tables when a partition has been dropped, and includes the following steps:

Step 1: Find the Low and High Range of the Partition that Was Dropped

When a partition is dropped, the range of the partition preceding it expands downwards. Therefore, there may be records in the preceding partition that should actually be in the dropped partition after it has been recovered. To ascertain this, run the following SQL script at the primary database, replacing the variables with the appropriate values:

```
SELECT * FROM partitioned_table
  WHERE relevant_key
  BETWEEN low_range_of_partition_that_was_dropped
  AND high_range_of_partition_that_was_dropped;
```

Step 2: Create a Temporary Table

If any records are returned, then create a temporary table in which to store these records so that if necessary they can be inserted into the recovered partition later.

Step 3: Delete Records From the Partitioned Table

Delete all the records stored in the temporary table from the partitioned table.

```
DELETE FROM partitioned_table
  WHERE relevant_key
  BETWEEN low_range_of_partition_that_was_dropped
  AND high_range_of_partition_that_was_dropped;
```

Step 4: Drop the Recovery Set Tablespace

On the primary database, drop each tablespace in the recovery set. For example, enter the following, replacing *tablespace_name* with the name of the tablespace:

```
DROP TABLESPACE tablespace_name INCLUDING CONTENTS;
```

Step 5: Create Tables at the Auxiliary Database

After opening the auxiliary database with the RESETLOGS option, create a table in the SYSTEM tablespace that has the same column names and column data types as the partitioned table you are recovering. You must create the table in the SYSTEM tablespace: otherwise, Oracle issues the ORA-01552 error. Create a table for each partition that you want to recover. These tables will be used later to swap each recovery set partition.

Step 6: Drop Indexes on Partitions Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, nonpartitioned indexes that exist on the partition you wish to recover.

Step 7: Exchange Partitions with Standalone Tables

For each partition in the auxiliary recovery set, exchange the partitions into the standalone tables created in "Step 5: Create Tables at the Auxiliary Database" on page 19-20 by issuing the following statement, replacing the variables with the appropriate values:

```
ALTER TABLE partitioned_table_name
EXCHANGE PARTITION partition_name
WITH TABLE table_name;
```

Step 8: Transport the Recovery Set Tablespaces

Export the recovery set tablespaces from the auxiliary database and then import them into the primary database as described in "Performing TSPITR with Transportable Tablespaces" on page 19-14.

Step 9: Insert Standalone Tables into Partitioned Tables

At this point you must insert the standalone tables into the partitioned tables; you can do this by first issuing the following statement, replacing the variables with the appropriate values:

```
ALTER TABLE table_name
  SPLIT PARTITION partition_name AT (key_value)
  INTO
  (PARTITION partition_1_name TABLESPACE tablespace_name,
   PARTITION partition_2_name TABLESPACE tablespace_name);
```

Aat this point, partition 2 is empty because keys in that range have already been deleted from the table.

Issue the following statement to swap the standalone table into the partition, replacing the variables with the appropriate values:

```
ALTER TABLE EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

Now insert the records saved in "Step 2: Create a Temporary Table" on page 19-20 into the recovered partition (if desired).

Note: If the partition that has been dropped is the last partition in the table, then add it with the ALTER TABLE ADD PARTITION statement.

Step 10: Back Up the Recovered Tablespaces in the Primary Database

Back up the recovered tablespaces in the primary database. Failure to do so results in loss of data in the event of media failure.

Performing TSPITR of Partitioned Tables When a Partition Has Split

This section describes how to recover partitioned tables when a partition has been split, and includes the following sections:

- Step 1: Drop the Lower of the Two Partitions at the Primary Database
- Steps 2: Follow Same Procedure as for Partial TSPITR of Partitioned Tablespaces

Step 1: Drop the Lower of the Two Partitions at the Primary Database

For each partition you wish to recover whose range has been split, drop the lower of the two partitions so that the higher expands downwards. In other words, the higher partition has the same range as before the split. For example, if P1 was split into partitions P1A and P1B, then P1B must be dropped, meaning that partition P1A now has the same range as P1.

For each partition that you wish to recover whose range has split, create a table that has exactly the same column names and column datatypes as the partitioned table you are recovering. For example, execute the following, replacing the variables with the appropriate values:

```
CREATE TABLE new_table
AS
(
  SELECT *
  FROM partitioned_table
  WHERE 1=2
);
```

These tables will be used to exchange each recovery set partition in "Step 3: Exchange Partitions with Standalone Tables" on page 19-17.

Steps 2: Follow Same Procedure as for Partial TSPITR of Partitioned Tablespaces

Follow the same procedure as for "Performing Partial TSPITR of Partitioned Tables" on page 19-16, but skip the first step of this procedure: "Step 1: Create a Table on the Primary Database for Each Partition Being Recovered" on page 19-17. In other words, start with "Step 2: Drop the Indexes on the Partition Being Recovered" on page 19-17 and follow all subsequent steps.

20

Troubleshooting User-Managed Media Recovery

This chapter describes how to troubleshoot user-managed media recovery, and includes the following topics:

About User-Managed Media Recovery Problems

Table 20–1 describes potential problems that can occur during media recovery.

Table 20–1 Media Recovery Problems

Problem	Description
Missing or misnamed archived log	Recovery stops because the database cannot find the archived log recorded in the control file.
When you attempt to open the database, error ORA-1113 indicates that a datafile needs media recovery	This error commonly occurs because: ■ You are performing incomplete recovery but failed to restore all needed datafile backups. ■ Incomplete recovery stopped before datafiles reached a consistent SCN. ■ You are recovering datafiles from an online backup, but not enough redo was applied to make the datafiles consistent. ■ You are performing recovery with a backup control file, and did not specify the location of a needed online redo log. ■ A datafile is undergoing media recovery when you attempt to open the database. ■ Datafiles needing recovery were not brought online before executing RECOVER DATABASE, and so were not recovered.
Redo record problems	Two possible cases are as follows: ■ Recovery stops because of failed consistency checks, a problem called **stuck recovery**. Stuck recovery can occur when an underlying operating system or storage system loses a write issued by the database during normal operation. ■ The database signals an internal error when applying the redo. This problem can be caused by an Oracle bug. If checksums are not being used, it can also be caused by corruptions to the redo or data blocks.
Corrupted archived logs	Logs may be corrupted while they are stored on or copied between storage systems. If DB_BLOCK_CHECKSUM is enabled, then the database usually signals checksum errors. If checksumming is not on, then log corruption may appear as a problem with redo.

Table 20–1 Media Recovery Problems (Cont.)

Problem	Description
Archived logs with incompatible parallel redo format	If you enable the parallel redo feature, then the database generates redo logs in a new format. Prior releases of Oracle are unable to apply parallel redo logs. However, releases prior to Oracle9*i* Release 2 (9.2) can detect the parallel redo format and indicate the inconsistency with the following error message: `External error 00303, 00000, "cannot process Parallel Redo"`. **See Also:** *Oracle Database Performance Tuning Guide* to learn about the parallel redo feature
Corrupted data blocks	A datafile backup may have contained a corrupted data block, or the data block may become corrupted either during recovery or when it was copied to the backup. If checksums are being used, then the database signals a checksum error. Otherwise, the problem may also appear as a redo corruption.
Random problems	Memory corruptions and other transient problems can occur during recovery.

The symptoms of media recovery problems are usually external or internal errors signaled during recovery. For example, an external error indicates that a redo block or a data block has failed checksum verification checks. Internal errors can be caused by either bugs in the database or errors arising from the underlying operating system and hardware.

If media recovery encounters a problem while recovering a database backup, whether it is a stuck recovery problem or a problem during redo application, the database always stops and leaves the datafiles undergoing recovery in a consistent state, that is, at a consistent SCN preceding the failure. You can then do one of the following:

- Open the database read-only to investigate the problem.
- Open the database with the `RESETLOGS` option, as long as the requirements for opening `RESETLOGS` have been met. Note that the `RESETLOGS` restrictions apply to opening the standby database as well, because a standby database is updated by a form of media recovery.

In general, opening the database read-only or opening with the `RESETLOGS` option require all online datafiles to be recovered to the same SCN. If this requirement is not met, then the database may signal `ORA-1113` or other errors when you attempt to open. Some common causes of `ORA-1113` are described in Table 20–1.

The basic methodology for responding to media recovery problems occurs in the following phases:

1. Try to identify the cause of the problem. Run a trial recovery if needed.

2. If the problem is related to missing redo logs or you suspect there is a redo log, memory, or data block corruption, then try to resolve it using the methods described in Table 20–2.
3. If you cannot resolve the problem using the methods described in Table 20–2, then do one of the following:
   - Open the database with the RESETLOGS option if you are recovering a whole database backup. If you have performed serial media recovery, then the database contains all the changes up to but not including the changes at the SCN where the corruption occurred. No changes from this SCN onward are in the recovered part of the database. If you have restored online backups, then opening RESETLOGS succeeds only if you have recovered through all the ALTER . . . END BACKUP operations in the redo stream.
   - Proceed with recovery by allowing media recovery to corrupt data blocks. After media recovery completes, try performing block media recovery using RMAN.
   - Call Oracle Support Services as a last resort.

     See Also: "Performing Block Media Recovery with RMAN" on page 8-21 to learn about block media recovery

Investigating the Media Recovery Problem: Phase 1

If media recovery encounters a problem, then obtain as much information as possible after recovery halts. You do not want to waste time fixing the wrong problem, which may in fact make matters worse.

The goal of this initial investigation is to determine whether the problem is caused by incorrect setup, corrupted redo logs, corrupted data blocks, memory corruption, or other problems. If you see a checksum error on a data block, then the data block is corrupted. If you see a checksum error on a redo log block, then the redo log is corrupted.

Sometimes the cause of a recovery problem can be difficult to determine. Nevertheless, the methods in this chapter allow you to quickly recover a database even when you do not completely understand the cause of the problem.

To investigate media recovery problems:

1. Examine the alert.log to see whether the error messages give general information about the nature of the problem. For example, does the alert_

SID.log indicate any checksum failures? Does the alert_*SID*.log indicate that media recovery may have to corrupt data blocks in order to continue?

2. Check the trace file generated by the Oracle process during recovery. It may contain additional error information.

Trying to Fix the Recovery Problem Without Corrupting Blocks: Phase 2

Depending on the type of media recovery problem you suspect, you have different solutions at your disposal. You can try one or a combination of the methods described in Table 20–2. Note that these methods are fairly safe: in almost all cases, they should not cause any damage to the database.

Table 20–2 Media Recovery Solutions

If you suspect . . .	Then . . .
Missing / misnamed archived logs	Determine whether you entered the correct filename. If you did, then check to see whether the log is missing from the operating system. If it is missing, and you have a backup, then restore the backup and apply the log. If you do not have a backup, then if possible perform incomplete recovery up to the point of the missing log.
ORA-1113 for ALTER DATABASE OPEN	Review the causes of this error in Table 20–1. Make sure that all read / write datafiles requiring recovery are online. If you use a backup control file for recovery, then the control file and datafiles must be at a consistent SCN for the database to be opened. If you do not have the necessary redo, then you must re-create the control file.
Corrupt archived logs	The log is corrupted if the checksum verification on the log redo block fails. If DB_BLOCK_CHECKSUM is not enabled either during the recovery session or when the database generated the redo, then recovery problems may be caused by corrupted logs. If the log is corrupt and an alternate copy of the corrupt log is available, then try to apply it and see whether this tactic fixes the problem. The DB_BLOCK_CHECKSUM initialization parameter determines whether checksums are computed for redo log and data blocks.

Table 20–2 Media Recovery Solutions (Cont.)

If you suspect . . .	Then . . .
Archived logs with incompatible parallel redo format	If you are running an Oracle release prior to Oracle9*i* Release 2, and if you are attempting to apply redo logs created with the parallel redo format, then you must do the following steps: **1.** Upgrade the database to a later release. **2.** Perform media recovery. **3.** Shut down the database consistently and back up the database. **4.** Downgrade the database to the original release. **See Also:** *Oracle Database Performance Tuning Guide* to learn about the parallel redo feature
Memory corruption or transient problems	You may be able to fix the problem by shutting down the database and restarting recovery. The databse should be left in a consistent state if the second attempt also fails.
Corrupt data blocks	Restore and recover the datafile again with user-managed methods, or restore and recover individual data blocks with the RMAN `BLOCKRECOVER` command. This tactic may fix the problem. A data block is corrupted if the checksum verification on the block fails. If `DB_BLOCK_CHECKING` is disabled, a corrupted data block problem may appear as a redo problem. If you must proceed with recovery, then you may want to corrupt the block now and continue recovery, and use RMAN to perform block media recovery later.

If you cannot fix the problem with the methods described in Table 20–2, then there may be no easy way to fix the problem without losing data. You have these options:

- Open the database with the `RESETLOGS` option (for whole database recovery). This solution discards all changes after the point where the redo problem occurred, but guarantees a logically consistent database.
- Allow media recovery to corrupt one or more data blocks and proceed with media recovery. This option will only succeed if the `alert_`*`SID`*`.log` indicates that recovery can continue if it is allowed to corrupt a data block, which should be the case for most recovery problems. This option is best if it is important to bring up the database quickly and recover all changes. If you are contemplating this option as a last resort, then proceed to "Deciding Whether to Allow Recovery to Corrupt Blocks: Phase 3" on page 20-7.

  See Also: "Performing Block Media Recovery with RMAN" on page 8-21 to learn how to perform block media recovery with the `BLOCKRECOVER` command

Deciding Whether to Allow Recovery to Corrupt Blocks: Phase 3

When media recovery encounters a problem, the `alert_SID.log` may indicate that recovery can continue if it is allowed to corrupt the data block causing the problem. The `alert_SID.log` always contains information about the block: its block type, block address, the tablespace it belongs to, and so forth. For blocks containing user data, the alert log may also report the data object number.

In this case, the database can proceed with recovery if it is allowed to mark the problem block as corrupt. Nevertheless, this response is not always advisable. For example, if the block is an important block in the SYSTEM tablespace, marking the block as corrupt can eventually prevent you from opening the recovered database. Another consideration is whether the recovery problem is isolated. If this problem is followed immediately by many other problems in the redo stream, then you may want to open the database with the RESETLOGS option.

For a block containing user data, you can usually query the database to find out which object or table owns this block. If the database is not open, then you should be able to open the database read-only, even if you are recovering a whole database backup. The following example cancels recovery and opens read-only:

```
CANCEL
ALTER DATABASE OPEN READ ONLY;
```

Assume that the data object number reported in the `alert_SID.log` is `8031`. You can determine the owner, object name, and object type by issuing this query:

```
SELECT OWNER, OBJECT_NAME, SUBOBJECT_NAME, OBJECT_TYPE
FROM DBA_OBJECTS
WHERE DATA_OBJECT_ID = 8031;
```

To determine whether a recovery problem is isolated, you can run a diagnostic **trial recovery**, which scans the redo stream for problems but does not actually make any changes to the recovered database. If a trial recovery discovers any recovery problems, it reports them in the `alert_SID.log`. You can use the RECOVER ... TEST statement to invoke trial recovery.

See Also: "Performing Trial Recovery" on page 20-9

After you have done these investigations, you can follow the guidelines in Table 20–3 to decide whether to allow recovery to corrupt blocks.

Table 20–3 Guidelines for Allowing Recovery to Permit Corruption

If the problem is . . .	and the block is . . .	Then . . .
not isolated	n/a	You should probably open the database with the RESETLOGS option. This response is important for stuck recovery problems, because stuck recovery can be caused by the operating system or a storage system losing writes. If an operating system or storage system suddenly fails, it can cause stuck recovery problems on several blocks.
isolated	in the SYSTEM tablespace	Do not corrupt the block, because it may eventually prevent you from opening the database. However, sometimes data in the SYSTEM tablespace is unimportant. If you must corrupt a SYSTEM block and recover all changes, contact Oracle Support.
isolated	index data	Consider corrupting index blocks because the index can be rebuilt later after the database has been recovered.
isolated	user data	Decide based on the importance of the data. If you continue with datafile recovery and corrupt a block, you lose data in the block. However, you can use RMAN to perform block media recovery later after datafile recovery completes. If you open RESETLOGS, then the database is consistent but loses any changes made after the point where recovery was stopped.
isolated	rollback or undo data	Consider corrupting the rollback or undo block because it does not harm the database if the transactions that generated the undo are never rolled back. However, if those transactions are rolled back, then corrupting the undo block can cause problems. If you are unsure, then call Oracle Support.

See Also: "Performing Trial Recovery" on page 20-9 to learn how to perform trial recovery, and "Allowing Recovery to Corrupt Blocks: Phase 4" on page 20-8 if you decide to corrupt blocks

Allowing Recovery to Corrupt Blocks: Phase 4

If you decide to allow recovery to proceed in spite of block corruptions, then run the RECOVER command with the ALLOW *n* CORRUPTION clause, where *n* is the number of allowable corrupt blocks.

To allow recovery to corrupt blocks:

1. Ensure that all normal recovery preconditions are met. For example, if the database is open, then take tablespaces offline before attempting recovery.

2. Run the RECOVER command, allowing a single corruption, repeating as necessary for each corruption to be made. The following statements shows a valid example:

```
RECOVER DATABASE ALLOW 1 CORRUPTION
```

Performing Trial Recovery

When problems such as stuck recovery occur, you have a difficult choice. If the block is relatively unimportant, and if the problem is isolated, then it is better to corrupt the block. But if the problem is not isolated, then it may be better to open the database with the RESETLOGS option.

Because of this situation, the Oracle database supports trial recovery. A trial recovery applies redo in a way similar to normal media recovery, but it never writes its changes to disk and it always rolls back its changes. Trial recovery occurs only in memory.

See Also: "Allowing Recovery to Corrupt Blocks: Phase 4" on page 20-8

How Trial Recovery Works

By default, if a trial recovery encounters a stuck recovery or similar problem, then it always marks the data block as corrupt in memory when this action can allow recovery to proceed. The database writes errors generated during trial recovery to alert files. These errors are clearly marked as test run errors.

Like normal media recovery, trial recovery can prompt you for archived log filenames and ask you to apply them. Trial recovery ends when:

- The database runs out of the maximum number of buffers in memory that trial recovery is permitted to use
- An unrecoverable error is signaled, that is, an error that cannot be resolved by corrupting a data block
- You cancel or interrupt the recovery session
- The next redo record in the redo stream changes the control file
- All requested redo has been applied

When trial recovery ends, the database removes all effects of the test run from the system—except the possible error messages in the alert files. If the instance fails

during trial recovery, then the database removes all effects of trial recovery from the system because trial recovery never writes changes to disk.

Trial recovery lets you foresee what problems might occur if you were to continue with normal recovery. For problems caused by ongoing memory corruption, trial recovery and normal recovery can encounter different errors.

Executing the RECOVER ... TEST Statement

You can use the TEST option for any RECOVER command. For example, you can start SQL*Plus and then issue any of the following commands:

```
RECOVER DATABASE TEST
RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL TEST
RECOVER TABLESPACE users TEST
RECOVER DATABASE UNTIL CANCEL TEST
```

By default, trial recovery always attempts to corrupt blocks in memory if this action allows trial recovery to proceed. In other words, trial recovery by default can corrupt an unlimited number of data blocks. You can specify the ALLOW *n* CORRUPTION clause on the RECOVER . . . TEST statement to limit the number of data blocks trial recovery can corrupt in memory.

A trial recovery command is usable in any scenario in which a normal recovery command is usable. Nevertheless, you should only need to run trial recovery when recovery runs into problems.

Index

A

B

C

D

E

F

M

N

O

P

R

S

T

U

V

W