

Oracle® Fusion Middleware

Administering Oracle GoldenGate



12c (12.3.0.1)

F12841-02

January 2019

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Fusion Middleware Administering Oracle GoldenGate, 12c (12.3.0.1)

F12841-02

Copyright © 2013, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	xvii
Documentation Accessibility	xvii
Related Information	xvii
Conventions	xviii

1 Oracle GoldenGate Administration Overview

2 Oracle GoldenGate Globalization Support

Preserving the Character Set	2-1
Character Set of Database Structural Metadata	2-1
Character Set of Character-type Data	2-1
Character Set of Database Connection	2-1
Character Set of Text Input and Output	2-2
Using Unicode and Native Characters	2-2

Part I Administering Oracle GoldenGate Classic Architecture

3 Configuring Manager and Network Communications

Overview of the Manager Process	3-1
Assigning Manager a Port for Local Communication	3-1
Maintaining Ports for Remote Connections through Firewalls	3-2
Choosing an Internet Protocol	3-3
Using the Recommended Manager Parameters	3-3
Creating the Manager Parameter File	3-3
Starting Manager	3-4
Starting Manager from the Command Shell of the Operating System	3-5
Starting Manager from GGSCI	3-5
Stopping Manager	3-5

Stopping Manager on UNIX and Linux	3-5
Stopping Manager on Windows	3-6

4 Getting Started with the Oracle GoldenGate Process Interfaces

Using the GGSCI Command-line Interface	4-1
Using Wildcards in Command Arguments	4-1
Globalization Support for the Command Interface	4-1
Using Command History	4-2
Storing and Calling Frequently Used Command Sequences	4-2
Controlling Oracle GoldenGate Processes	4-3
Controlling Manager	4-3
Controlling Extract and Replicat	4-4
Deleting Extract and Replicat	4-4
Automating Commands	4-5
Issuing Commands Through the IBM i CLI	4-6
Using Oracle GoldenGate Parameter Files	4-7
Globalization Support for Parameter Files	4-7
Working with the GLOBALS File	4-7
Working with Runtime Parameters	4-8
Creating a Parameter File	4-10
Creating a Parameter File in GGSCI	4-10
Creating a Parameter File with a Text Editor	4-12
Validating a Parameter File	4-12
Viewing a Parameter File	4-15
Changing a Parameter File	4-16
Simplifying the Creation of Parameter Files	4-16
Using Wildcards	4-17
Using OBEY	4-17
Using Macros	4-17
Using Parameter Substitution	4-17
Getting Information about Oracle GoldenGate Parameters	4-18
Specifying Object Names in Oracle GoldenGate Input	4-19
Specifying Filesystem Path Names in Parameter Files on Windows Systems	4-19
Supported Database Object Names	4-19
Supported Special Characters	4-20
Non-supported Special Characters	4-20
Specifying Names that Contain Slashes	4-21
Qualifying Database Object Names	4-21
Two-part Names	4-21
Three-part Names	4-22

Applying Data from Multiple Containers or Catalogs	4-22
Specifying a Default Container or Catalog	4-22
Specifying Case-Sensitive Database Object Names	4-23
Using Wildcards in Database Object Names	4-24
Rules for Using Wildcards for Source Objects	4-25
Rules for Using Wildcards for Target Objects	4-26
Fallback Name Mapping	4-26
Wildcard Mapping from Pre-11.2.1 Trail Version	4-26
Asterisks or Question Marks as Literals in Object Names	4-27
How Wildcards are Resolved	4-27
Excluding Objects from a Wildcard Specification	4-27
Differentiating Case-Sensitive Column Names from Literals	4-27

5 Using Oracle GoldenGate for Live Reporting

Overview of the Reporting Configuration	5-1
Filtering and Conversion	5-1
Read-only vs. High Availability	5-2
Additional Information	5-2
Creating a Standard Reporting Configuration	5-2
Source System	5-3
Target System	5-4
Creating a Reporting Configuration with a Data Pump on the Source System	5-5
Source System	5-5
Target System	5-7
Creating a Reporting Configuration with a Data Pump on an Intermediary System	5-8
Source System	5-10
Intermediary System	5-11
Target System	5-12
Creating a Cascading Reporting Configuration	5-13
Source System	5-16
Second System in the Cascade	5-18
Third System in the Cascade	5-20

6 Using Oracle GoldenGate for Real-time Data Distribution

Overview of the Data-distribution Configuration	6-1
Considerations for a Data-distribution Configuration	6-1
Fault Tolerance	6-2
Filtering and Conversion	6-2
Read-only vs. High Availability	6-2

Additional Information	6-2
Creating a Data Distribution Configuration	6-2
Source System	6-3
Target Systems	6-5

7 Configuring Oracle GoldenGate for Real-time Data Warehousing

Overview of the Data Warehousing Configuration	7-1
Considerations for a Data Warehousing Configuration	7-1
Isolation of Data Records	7-2
Data Storage	7-2
Filtering and Conversion	7-2
Additional Information	7-2
Creating a Data Warehousing Configuration	7-2
Source Systems	7-3
Target System	7-6

8 Configuring Oracle GoldenGate to Maintain a Live Standby Database

Overview of a Live Standby Configuration	8-1
Considerations for a Live Standby Configuration	8-2
Trusted Source	8-3
Duplicate Standby	8-3
DML on the Standby System	8-3
Oracle GoldenGate Processes	8-3
Backup Files	8-3
Failover Preparedness	8-3
Sequential Values that are Generated by the Database	8-4
Additional Information	8-4
Creating a Live Standby Configuration	8-4
Prerequisites on Both Systems	8-5
Configuration from Active Source to Standby	8-5
Configuration from Standby to Active Source	8-7
Moving User Activity in a Planned Switchover	8-10
Moving User Activity to the Live Standby	8-10
Moving User Activity Back to the Primary System	8-11
Moving User Activity in an Unplanned Failover	8-13
Moving User Activity to the Live Standby	8-13
Moving User Activity Back to the Primary System	8-13

9 Configuring Oracle GoldenGate for Active-Active High Availability

Overview of an Active-Active Configuration	9-1
Considerations for an Active-Active Configuration	9-2
TRUNCATES	9-2
Application Design	9-2
Keys	9-3
Triggers and Cascaded Deletes	9-3
Database-Generated Values	9-4
Database Configuration	9-4
Preventing Data Looping	9-4
Preventing the Capture of Replicat Operations	9-4
Preventing the Capture of Replicat Transactions (Oracle)	9-5
Preventing Capture of Replicat Transactions (Other Databases)	9-5
Identifying Replicat Transactions	9-5
DB2 z/OS, DB2 LUW, and DB2 for i	9-5
MySQL	9-6
Oracle	9-6
SQL Server	9-6
Replicating DDL in a Bi-directional Configuration	9-7
Managing Conflicts	9-7
Additional Information	9-8
Creating an Active-Active Configuration	9-8
Prerequisites on Both Systems	9-9
Configuration from Primary System to Secondary System	9-9
Configuration from Secondary System to Primary System	9-12

10 Configuring Conflict Detection and Resolution

Overview of the Oracle GoldenGate CDR Feature	10-1
Configuring Oracle GoldenGate CDR	10-2
Making the Required Column Values Available to Extract	10-2
Configuring the Oracle GoldenGate Parameter Files for Conflict Resolution	10-2
Configuring the Oracle GoldenGate Parameter Files for Error Handling	10-3
Tools for Mapping Extra Data to the Exceptions Table	10-4
Sample Exceptions Mapping with Source and Target Columns Only	10-5
Sample Exceptions Mapping with Additional Columns in the Exceptions Table	10-6
Viewing CDR Statistics	10-8
Report File	10-8
GGSCI	10-8
Column-conversion Functions	10-8

CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD	10-9
Table Used in this Example	10-9
MAP Statement with Conflict Resolution Specifications	10-9
Description of MAP Statement	10-10
Error Handling	10-10
INSERTROWEXISTS with the USEMAX Resolution	10-10
UPDATEROWEXISTS with the USEMAX Resolution	10-11
UPDATEROWMISSING with OVERWRITE Resolution	10-12
DELETEROWMISSING with DISCARD Resolution	10-13
DELETEROWEXISTS with OVERWRITE Resolution	10-14
CDR Example 2: UPDATEROWEXISTS with USEDELTA and USEMAX	10-15
Table Used in this Example	10-15
MAP Statement	10-16
Description of MAP Statement	10-16
Error Handling	10-16
CDR Example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE	10-18
Table Used in this Example	10-18
MAP Statement	10-18
Description of MAP Statement	10-18
Error Handling	10-19

11 Mapping and Manipulating Data

Limitations of Support	11-1
Parameters that Control Mapping and Data Integration	11-1
Mapping between Dissimilar Databases	11-2
Deciding Where Data Mapping and Conversion Will Take Place	11-2
Mapping and Conversion on Windows and UNIX Systems	11-2
Mapping and Conversion on NonStop Systems	11-2
Globalization Considerations when Mapping Data	11-2
Conversion between Character Sets	11-3
Database Object Names	11-3
Column Data	11-3
Preservation of Locale	11-4
Support for Escape Sequences	11-4
Mapping Columns	11-6
Supporting Case and Special Characters in Column Names	11-6
Configuring Table-level Column Mapping with COLMAP	11-6
Specifying the Columns to be Mapped in the COLMAP Clause	11-6
Using USEDEFAULTS to Enable Default Column Mapping	11-8
Determining Whether COLMAP Requires a Data-definitions File	11-9

Configuring Global Column Mapping with COLMATCH	11-10
Understanding Default Column Mapping	11-13
Mapping Data Types from Column to Column	11-13
Numeric Columns	11-13
Character-type Columns	11-14
Datetime Columns	11-14
Selecting and Filtering Rows	11-14
Selecting Rows with a FILTER Clause	11-14
Selecting Rows with a WHERE Clause	11-17
Considerations for Selecting Rows with FILTER and WHERE	11-18
Ensuring Data Availability for Filters	11-18
Comparing Column Values	11-19
Testing for NULL Values	11-19
Retrieving Before and After Values	11-19
Selecting Columns	11-20
Selecting and Converting SQL Operations	11-20
Using Transaction History	11-21
Testing and Transforming Data	11-22
Handling Column Names and Literals in Functions	11-24
Using the Appropriate Function	11-24
Transforming Dates	11-24
Performing Arithmetic Operations	11-24
Omitting @COMPUTE	11-25
Manipulating Numbers and Character Strings	11-25
Handling Null, Invalid, and Missing Data	11-26
Using @COLSTAT	11-26
Using @COLTEST	11-27
Using @IF	11-27
Performing Tests	11-27
Using @CASE	11-27
Using @VALONEOF	11-28
Using @EVAL	11-28
Using Tokens	11-28
Defining Tokens	11-28
Using Token Data in Target Tables	11-29

12 Associating Replicated Data with Metadata

Overview	12-1
Understanding Data Definition Files	12-1
Contents of the Definitions File	12-2

Which Definitions File Type to Use, and Where	12-2
Understanding the Effect of Character Sets on Definitions Files	12-3
Confining Data Mapping and Conversion to the Replicat Process	12-3
Avoiding File Corruptions Due to Operating System Character Sets	12-3
Changing the Character Set of Existing Definitions Files	12-3
Downloading from a z/OS system to another platform	12-4
Using a Definitions Template	12-4
Configuring Oracle GoldenGate to Capture Data-definitions	12-4
Configure DEFGEN	12-4
Run DEFGEN	12-6
Transfer the Definitions File to the Remote System	12-7
Specify the Definitions File	12-7
Adding Tables that Satisfy a Definitions Template	12-7
Examples of Using a Definitions File	12-7
Creating a Source-definitions file for Use on a Target System	12-8
Creating Target-definitions Files for Use on a Source System	12-8
Creating Multiple Source Definition Files for Use on a Target System	12-9
Using Automatic Trail File Recovery	12-10
Configuring Oracle GoldenGate to Use Self-Describing Trail Files	12-10
Support Considerations	12-12
Using Self-Describing Trail Files	12-12
Examples of Parameter Files	12-13
Configuring Oracle GoldenGate to Assume Identical Metadata	12-14
Rules for Tables to be Considered Identical	12-14
Configuring Oracle GoldenGate to Assume Dissimilar Metadata	12-15
Configuring Oracle GoldenGate to Use a Combination of Similar and Dissimilar Definitions	12-15

13 Configuring Online Change Synchronization

Overview of Online Change Synchronization	13-1
Initial Synchronization	13-2
Choosing Names for Processes and Files	13-2
Naming Conventions for Processes	13-2
Choosing File Names	13-3
Creating a Checkpoint Table	13-4
Options for Creating the Checkpoint Table	13-4
Adjusting for Coordinated Replicat in Oracle RAC	13-5
Creating an Online Extract Group	13-6
Creating a Trail	13-8
Assigning Storage for Oracle GoldenGate Trails	13-9
Estimating Space for the Trails	13-9

Adding a Trail	13-10
Creating a Parameter File for Online Extraction	13-10
Creating an Online Replicat Group	13-12
About Classic Replicat Mode	13-13
About Coordinated Replicat Mode	13-14
About Barrier Transactions	13-15
How Barrier Transactions are Processed	13-16
About the Global Watermark	13-16
About Integrated Replicat Mode	13-17
Understanding Replicat Processing in Relation to Parameter Changes	13-17
Creating the Replicat Group	13-17
Creating a Parameter File for Online Replication	13-19

14 Handling Processing Errors

Overview of Oracle GoldenGate Error Handling	14-1
Handling Extract Errors	14-1
Handling Replicat Errors during DML Operations	14-2
Handling Errors as Exceptions	14-2
Using EXCEPTIONSONLY	14-3
Using MAPEXCEPTION	14-4
About the Exceptions Table	14-5
Handling Replicat errors during DDL Operations	14-5
Handling TCP/IP Errors	14-5
Maintaining Updated Error Messages	14-6
Resolving Oracle GoldenGate Errors	14-6

15 Instantiating Oracle GoldenGate with an Initial Load

Overview of the Initial-Load Procedure	15-1
Improving the Performance of an Initial Load	15-1
Prerequisites for Initial Load	15-1
Disable DDL Processing	15-2
Prepare the Target Tables	15-2
Configure the Manager Process	15-2
Create a Data-definitions File	15-3
Create Change-synchronization Groups	15-3
Sharing Parameters between Process Groups	15-3
Initial Load in Classic Architecture	15-3
Loading Data with a Database Utility	15-4
Loading Data with Oracle Data Pump	15-6

Using Automatic Per Table Instantiation	15-6
Using Oracle Data Pump Table Instantiation	15-7
Loading Data from File to Replicat	15-7
Loading Data with an Oracle GoldenGate Direct Load	15-13
Loading Data with a Direct Bulk Load to SQL*Loader	15-18
Loading Data with Teradata Load Utilities	15-23

16 Customizing Oracle GoldenGate Processing

Executing Commands, Stored Procedures, and Queries with SQLEXEC	16-1
Performing Processing with SQLEXEC	16-2
Using SQLEXEC	16-2
Executing SQLEXEC within a TABLE or MAP Statement	16-2
Executing SQLEXEC as a Standalone Statement	16-3
Using Input and Output Parameters	16-4
Passing Values to Input Parameters	16-4
Passing Values to Output Parameters	16-5
SQLEXEC Examples Using Parameters	16-5
Handling SQLEXEC Errors	16-6
Handling Missing Column Values	16-7
Handling Database Errors	16-7
Additional SQLEXEC Guidelines	16-7
Using Oracle GoldenGate Macros to Simplify and Automate Work	16-8
Defining a Macro	16-9
Calling a Macro	16-10
Calling a Macro that Contains Parameters	16-12
Calling a Macro without Input Parameters	16-13
Calling Other Macros from a Macro	16-14
Creating Macro Libraries	16-14
Tracing Macro Expansion	16-16
Using User Exits to Extend Oracle GoldenGate Capabilities	16-16
When to Implement User Exits	16-17
Making Oracle GoldenGate Record Information Available to the Routine	16-17
Creating User Exits	16-17
Supporting Character-set Conversion in User Exits	16-19
Using Macros to Check Name Metadata	16-19
Describing the Character Format	16-20
Upgrading User Exits	16-22
Viewing Examples of How to Use the User Exit Functions	16-22
Using the Oracle GoldenGate Event Marker System to Raise Database Events	16-23
Case Studies in the Usage of the Event Marker System	16-24

Trigger End-of-day Processing	16-24
Simplify Transition from Initial Load to Change Synchronization	16-24
Stop Processing When Data Anomalies are Encountered	16-25
Trace a Specific Order Number	16-25
Execute a Batch Process	16-25
Propagate Only a SQL Statement without the Resultant Operations	16-26
Committing Other Transactions Before Starting a Long-running Transaction	16-26
Execute a Shell Script to Validate Data	16-26

17 Monitoring Oracle GoldenGate Processing

Using the Information Commands in GGSCI	17-1
Monitoring an Extract Recovery	17-2
Monitoring Lag	17-3
About Lag	17-3
Controlling How Lag is Reported	17-3
Using Automatic Heartbeat Tables to Monitor	17-4
Understanding Heartbeat Table End-To-End Replication Flow	17-5
Updating Heartbeat Tables	17-12
Purging the Heartbeat History Tables	17-12
Best Practice	17-12
Using the Automatic Heartbeat Commands	17-13
Monitoring Processing Volume	17-13
Using the Error Log	17-13
Using the Process Report	17-14
Scheduling Runtime Statistics in the Process Report	17-15
Viewing Record Counts in the Process Report	17-15
Preventing SQL Errors from Filling the Replicat Report File	17-15
Using the Discard File	17-15
Maintaining the Discard and Report Files	17-16
Reconciling Time Differences	17-17
Getting Help with Performance Tuning	17-17

18 Tuning the Performance of Oracle GoldenGate

Using Multiple Process Groups	18-1
Considerations for Using Multiple Process Groups	18-2
Maintaining Data Integrity	18-3
Number of Groups	18-3
Memory	18-3
Isolating Processing-Intensive Tables	18-4

Using Parallel Replicat Groups on a Target System	18-4
To Create the Extract Group	18-4
To Create the Replicat Groups	18-5
Using Multiple Extract Groups with Multiple Replicat Groups	18-5
To Create the Extract Groups	18-6
To Create the Replicat Groups	18-6
Splitting Large Tables Into Row Ranges Across Process Groups	18-6
Configuring Oracle GoldenGate to Use the Network Efficiently	18-7
Detecting a Network Bottleneck that is Affecting Oracle GoldenGate	18-8
Working Around Bandwidth Limitations by Using Data Pumps	18-9
Reducing the Bandwidth Requirements of Oracle GoldenGate	18-9
Increasing the TCP/IP Packet Size	18-9
Eliminating Disk I/O Bottlenecks	18-10
Improving I/O performance Within the System Configuration	18-10
Improving I/O Performance Within the Oracle GoldenGate Configuration	18-11
Managing Virtual Memory and Paging	18-11
Optimizing Data Filtering and Conversion	18-12
Tuning Replicat Transactions	18-13
Tuning Coordination Performance Against Barrier Transactions	18-13
Applying Similar SQL Statements in Arrays	18-13
Preventing Full Table Scans in the Absence of Keys	18-14
Splitting Large Transactions	18-14
Adjusting Open Cursors	18-14
Improving Update Speed	18-15
Set a Replicat Transaction Timeout	18-15

19 Performing Administrative Operations

Performing Application Patches	19-1
Initializing the Transaction Logs	19-2
Shutting Down the System	19-4
Changing Database Attributes	19-4
Changing Database Metadata	19-4
Adding Tables to the Oracle GoldenGate Configuration	19-6
Coordinating Table Attributes between Source and Target	19-7
Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables	19-9
Dropping and Recreating a Source Table	19-9
Changing the Number of Oracle RAC Threads when Using Classic Capture	19-10
Changing the ORACLE_SID	19-11
Purging Archive Logs	19-11
Reorganizing a DB2 Table (z/OS Platform)	19-12

Adding Process Groups to an Active Configuration	19-12
Before You Start	19-12
Adding Another Extract Group to an Active Configuration	19-12
Adding Another Data Pump to an Active Configuration	19-15
Adding Another Replicat Group to an Active Configuration	19-17
Changing the Size of Trail Files	19-19
Switching Extract from Classic Mode to Integrated Mode	19-19
Switching Extract from Integrated Mode to Classic Mode	19-20
Switching Replicat from Nonintegrated Mode to Integrated Mode	19-22
Switching Replicat from Integrated Mode to Nonintegrated Mode	19-23
Switching Replicat to Coordinated Mode	19-24
Procedure Overview	19-24
Performing the Switch to Coordinated Replicat	19-25
Administering a Coordinated Replicat Configuration	19-27
Performing a Planned Re-partitioning of the Workload	19-27
Recovering Replicat After an Unplanned Re-partitioning	19-28
Reprocessing From the Low Watermark with HANDLECOLLISIONS	19-28
Using the Auto-Saved Parameter File	19-29
Synchronizing Threads After an Unclean Stop	19-30
Restarting a Primary Extract after System Failure or Corruption	19-30
Details of This Procedure	19-31
Performing the Recovery	19-31

Part II Administering Oracle GoldenGate Microservices Architecture

20 Loading Data from File to Replicat in Microservices Architecture

A Supported Character Sets

Supported Character Sets - Oracle	A-1
Supported Character Sets - Non-Oracle	A-8

B Supported Locales

C About the Oracle GoldenGate Trail

Trail Recovery Mode	C-1
Trail File Header Record	C-1

Trail Record Format	C-2
Example of an Oracle GoldenGate Record	C-2
Record Header Area	C-3
Description of Header Fields	C-3
Using Header Data	C-5
Record Data Area	C-5
Full Record Image Format (NonStop Sources)	C-6
Compressed Record Image Format (Windows, UNIX, Linux Sources)	C-6
Tokens Area	C-7
Oracle GoldenGate Operation Types	C-7
Oracle GoldenGate Trail Header Record	C-10

D Using the Commit Sequence Number

E About Checkpoints

Extract Checkpoints	E-1
About Extract read checkpoints	E-3
Startup Checkpoint	E-3
Recovery Checkpoint	E-3
Current Checkpoint	E-3
About Extract Write Checkpoints	E-4
Replicat Checkpoints	E-4
About Replicat Checkpoints	E-5
Startup Checkpoint	E-5
Current Checkpoint	E-5
Internal Checkpoint Information	E-5
Oracle GoldenGate Checkpoint Tables	E-6

Preface

This guide contains instructions for:

- Working with the interface components that control Oracle GoldenGate.
- Monitoring and troubleshooting Oracle GoldenGate performance.
- Perform other administrative operations.
- [Audience](#)
- [Documentation Accessibility](#)
- [Related Information](#)
- [Conventions](#)

Audience

This guide is intended for the person or persons who are responsible for operating Oracle GoldenGate and maintaining its performance. This audience typically includes, but is not limited to, systems administrators and database administrators.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Information

The Oracle GoldenGate Product Documentation Libraries are found at

<https://docs.oracle.com/en/middleware/goldengate/index.html>

Additional Oracle GoldenGate information, including best practices, articles, and solutions, is found at:

[Oracle GoldenGate A-Team Chronicles](#)

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, such as "From the File menu, select Save ." Boldface also is used for terms defined in text or in the glossary.
<i>italic</i>	Italic type indicates placeholder variables for which you supply particular values, such as in the parameter statement: <code>TABLE <i>table_name</i></code> . Italic type also is used for book titles and emphasis.
<i>italic</i>	
monospace MONOSPACE	Monospace type indicates code components such as user exits and scripts; the names of files and database objects; URL paths; and input and output text that appears on the screen. Uppercase monospace type is generally used to represent the names of Oracle GoldenGate parameters, commands, and user-configurable functions, as well as SQL commands and keywords.
UPPERCASE	Uppercase in the regular text font indicates the name of a utility unless the name is intended to be a specific case.
{ }	Braces within syntax enclose a set of options that are separated by pipe symbols, one of which must be selected, for example: <code>{<i>option1</i> <i>option2</i> <i>option3</i>}</code> .
[]	Brackets within syntax indicate an optional element. For example in this syntax, the <code>SAVE</code> clause is optional: <code>CLEANUP REPLICAT <i>group_name</i> [, <i>SAVE count</i>]</code> . Multiple options within an optional element are separated by a pipe symbol, for example: <code>[<i>option1</i> <i>option2</i>]</code> .

1

Oracle GoldenGate Administration Overview

Administrative decisions for Oracle GoldenGate processes depend on the type of architecture you are using.

As an administrator for Oracle GoldenGate, you need to ensure that the configuration and processes are relevant to the type of architecture that's implemented in your production environment. This book is divided into two parts to describe the configurations, processes, and tasks that are specific to the Classic Architecture and the Microservices Architecture.

See *Getting Started with the Oracle GoldenGate Architectures* for conceptual details about the Oracle GoldenGate architectures.

This book is divided into two parts to describe processes that are specific to each architecture:

- [Part 1: Administering Oracle GoldenGate Classic Architecture](#)
- [Part 2: Administering Oracle GoldenGate Microservices Architecture](#)

 **Note:**

Oracle GoldenGate 18c (18.1.0) is not released for SQL Server and DB2 for i. However, the documentation may include information associated with these databases.

2

Oracle GoldenGate Globalization Support

This chapter describes Oracle GoldenGate globalization support, which enables the processing of data in its native language encoding.

Topics:

- [Preserving the Character Set](#)
- [Using Unicode and Native Characters](#)

Preserving the Character Set

In order to process the data in its native language encoding, Oracle GoldenGate takes into consideration the character set of the database and the operating system locale, if applicable.

- [Character Set of Database Structural Metadata](#)
- [Character Set of Character-type Data](#)
- [Character Set of Database Connection](#)
- [Character Set of Text Input and Output](#)

Character Set of Database Structural Metadata

Oracle GoldenGate processes catalog, schema, table and column names in their native language as determined by the character set encoding of the source and target databases. This processing is extended to the parameter files and command interpreter, where they are processed according to the operating system locale. These objects appear in their localized format throughout the client interface, on the console, and in files.

Character Set of Character-type Data

The Oracle GoldenGate apply process (Replicat) supports the conversion of data from one character set to another when the data is contained in character column types. Character-set conversion support is limited to column-to-column mapping as performed with the `COLMAP` or `USEDEFAULTS` clauses of a `TABLE` or `MAP` statement. It is not supported by the column-conversion functions, by `SQLEXEC`, or by the `TOKENS` feature.

See [Mapping and Manipulating Data](#) for more information about character sets, conversion between them, and data mapping.

Character Set of Database Connection

The Extract and Replicat processes use a session character set when connecting to the database. For Oracle Databases, the session character set is set to the same as the database character set by both Extract and Replicat. For MySQL, the session

character set is taken from the `SESSIONCHARSET` option of `SOURCEDB` and `TARGETDB`, or from the `SESSIONCHARSET` parameter set globally in the `GLOBALS` file. For other database types, it is obtained programmatically. In addition, Oracle GoldenGate processes use a session character set for communication and data transfer between Oracle GoldenGate and the database, such as for SQL queries, fetches, and applying data.

Character Set of Text Input and Output

Oracle GoldenGate supports text input and output in the default character set of the host operating system for the following:

- Console
- Command-line input and output
- `FORMATASCII`, `FORMATSQL`, `FORMATXML` parameters, text files such as parameter files, data-definitions files, error log, process reports, discard files, and other human-readable files that are used by Oracle GoldenGate users to configure, run, and monitor the Oracle GoldenGate environment.

In the event that the platform does not support a required character set as the default in the operating system, you can use the following parameters to specify a character set:

- `CHARSET` parameter to specify a character set to be used by processes to read their parameter files.
- `CHARSET` option of the `DEFSFILE` parameter to generate a data-definitions file in a specific character set.

The `GGSCI` command console always operates in the character set of the local operating system for both keyboard and `OBEY` file input and console output.

Using Unicode and Native Characters

Oracle GoldenGate supports the use of an escape sequence to represent characters in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. You can use an escape sequence if the operating system does not support the required character, or for any other purpose when needed. For more information about this support, see [Support for Escape Sequences](#).

Part I

Administering Oracle GoldenGate Classic Architecture

Oracle GoldenGate Classic Architecture provides the processes and files required to effectively move data across a variety of topologies. These processes and files form the main components of the classic architecture and was the product design until this release.

- [Configuring Manager and Network Communications](#)
- [Getting Started with the Oracle GoldenGate Process Interfaces](#)
- [Using Oracle GoldenGate for Live Reporting](#)
- [Using Oracle GoldenGate for Real-time Data Distribution](#)
- [Configuring Oracle GoldenGate for Real-time Data Warehousing](#)
- [Configuring Oracle GoldenGate to Maintain a Live Standby Database](#)
- [Configuring Oracle GoldenGate for Active-Active High Availability](#)
- [Configuring Conflict Detection and Resolution](#)
- [Mapping and Manipulating Data](#)
- [Associating Replicated Data with Metadata](#)
- [Configuring Online Change Synchronization](#)
- [Handling Processing Errors](#)
- [Instantiating Oracle GoldenGate with an Initial Load](#)
- [Customizing Oracle GoldenGate Processing](#)
- [Monitoring Oracle GoldenGate Processing](#)
- [Tuning the Performance of Oracle GoldenGate](#)
- [Performing Administrative Operations](#)

3

Configuring Manager and Network Communications

This chapter describes how to configure the Manager process and specify ports for local and remote network communications.

Topics:

- [Overview of the Manager Process](#)
- [Assigning Manager a Port for Local Communication](#)
- [Maintaining Ports for Remote Connections through Firewalls](#)
- [Choosing an Internet Protocol](#)
- [Using the Recommended Manager Parameters](#)
- [Creating the Manager Parameter File](#)
- [Starting Manager](#)
- [Stopping Manager](#)

Overview of the Manager Process

To configure and run Oracle GoldenGate, a Manager process must be running on all Oracle GoldenGate source and target systems, and any intermediary systems if used in your configuration. Manager is the controller process that instantiates the Oracle GoldenGate processes, allocates port numbers, and performs file maintenance. Together, the Manager process and its child processes, and their related programs and files comprise an Oracle GoldenGate instance. The Manager process performs the following functions:

- Start Oracle GoldenGate processes
- Start dynamic processes
- Start the Collector process
- Manage the port numbers for processes. (All Oracle GoldenGate ports are configurable.)
- Perform trail management
- Create event, error, and threshold reports

There is one Manager per Oracle GoldenGate installation. One Manager can support multiple Oracle GoldenGate extraction and replication processes.

Assigning Manager a Port for Local Communication

The Manager process in each Oracle GoldenGate installation requires a dedicated port for communication between itself and other local Oracle GoldenGate processes.

To specify this port, use the `PORT` parameter in the Manager parameter file. Follow these guidelines:

- The default port number for Manager is 7809. You must specify either the default port number (recommended, if available) or a different one of your choice.
- The port must be unreserved and unrestricted.
- Each Manager instance on a system must use a different port number.

See `PORT` in *Reference for Oracle GoldenGate* for more information.

Maintaining Ports for Remote Connections through Firewalls

If a firewall is being used at an Oracle GoldenGate target location, additional ports are required on the target system to receive dynamic TCP/IP communications from remote Oracle GoldenGate processes. These ports are:

- One port for *each* Collector process that is started by the local Manager to receive propagated transaction data from remote online Extract processes. When an Extract process sends data to a target, the Manager on the target starts a dedicated Collector process.
- One port for *each* Replicat process that is started by the local Manager as part of a remote task. A remote task is used for initial loads and is specified with the `RMTTASK` parameter. This port is used to receive incoming requests from the remote Extract process. See `RMTTASK` in *Reference for Oracle GoldenGate* for more information.
- Some extra ports in case they are needed for expansion of the local Oracle GoldenGate configuration.
- Ports for the other Oracle GoldenGate products if they interact with the local Oracle GoldenGate instance, as stated in the documentation of those products.

To specify these ports, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Follow these guidelines:

- You can specify up to 5000 ports in any combination of the following formats:

```
7830, 7833, 7835  
7830-7835  
7830-7835, 7839
```

- The ports must be unreserved and unrestricted.
- Each Manager instance on a system must use a different port list.

Although not a required parameter, `DYNAMICPORTLIST` is strongly recommended for best performance. The Collector process is responsible for finding and binding to an available port, and having a known list of qualified ports speeds this process. In the absence of `DYNAMICPORTLIST` (or if not enough ports are specified with it), Collector tries to use port 7840 for remote requests. If 7840 is not available, Collector increments by one until it finds an available port. This can delay the acceptance of the remote request. If Collector runs out of ports in the `DYNAMICPORTLIST` list, the following occurs:

- Manager reports an error in its process report and in the Oracle GoldenGate `ggserr` log.

- Collector retries based on the rules in the Oracle GoldenGate `tcperrs` file. For more information about the `tcperrs` file, see [Handling TCP/IP Errors](#).

See `DYNAMICPORTLIST` in *Reference for Oracle GoldenGate* for more information.

Choosing an Internet Protocol

By default, Oracle GoldenGate selects a socket in the following order of priority to ensure the best chance of connection success:

- IPv6 dual-stack
- IPv4 if IPv6 dual-stack is not available
- IPv6

If your network has IPv6 network devices that do not support dual-stack mode, you can use the `USEIPV6` parameter to force Oracle GoldenGate to use IPv6 for all connections. This is a `GLOBALS` parameter that applies to all processes of an Oracle GoldenGate instance. When `USEIPV6` is used, the entire network must be IPv6 compatible to avoid connection failures. See `USEIPV6` in *Reference for Oracle GoldenGate* for more information.

Using the Recommended Manager Parameters

The following parameters are optional, but recommended, for the Manager process.

- `AUTOSTART`: Starts Extract and Replicat processes when Manager starts. This parameter is required in a cluster configuration, and is useful when Oracle GoldenGate activities must begin immediately at system startup. (Requires Manager to be part of the startup routine.) You can use multiple `AUTOSTART` statements in the same parameter file. See `AUTOSTART` in *Reference for Oracle GoldenGate* for more information.
- `AUTORESTART`: Starts Extract and Replicat processes again after abnormal termination. This parameter is required in a cluster configuration, but is also useful in any configuration to ensure continued processing. See `AUTORESTART` in *Reference for Oracle GoldenGate* for more information.
- `PURGEOLDEXTRACTS`: Purges trail files when Oracle GoldenGate is finished processing them. Without `PURGEOLDEXTRACTS`, no purging is performed and trail files can consume significant disk space. For best results, use `PURGEOLDEXTRACTS` as a Manager parameter, not as an Extract or Replicat parameter. See `PURGEOLDEXTRACTS` in *Reference for Oracle GoldenGate* for more information.
- `STARTUPVALIDATIONDELAY` | `STARTUPVALIDATIONDELAYCSECS`: Sets a delay time after which Manager validates the run status of a process. Startup validation makes Oracle GoldenGate users aware of processes that fail before they can generate an error message or process report. See `STARTUPVALIDATIONDELAYCSECS` in *Reference for Oracle GoldenGate* for more information.

Creating the Manager Parameter File

To configure Manager with required port information and optional parameters, create a parameter file by following these steps. See [Getting Started with the Oracle](#)

[GoldenGate Process Interfaces](#)" for more information about Oracle GoldenGate parameter files.

 **Note:**

If Oracle GoldenGate resides in a cluster, configure the Manager process within the cluster application as directed by the vendor's documentation, so that Oracle GoldenGate fails over properly with other applications. For more information about installing Oracle GoldenGate in a cluster, see the Oracle GoldenGate documentation for your database.

1. From the Oracle GoldenGate directory, run the GGSCI program to open the Oracle GoldenGate Software Command Interface (GGSCI).
2. In GGSCI, issue the following command to edit the Manager parameter file.

```
EDIT PARAMS MGR
```
3. Add the parameters that you want to use for the Manager process, each on one line.
4. Save, then close the file.

Example 3-1 Sample manager file on a UNIX system

```
PORT 7809
DYNAMICPORTLIST 7810-7820, 7830
AUTOSTART ER t*
AUORESTART ER t*, RETRIES 4, WAITMINUTES 4
STARTUPVALIDATIONDELAY 5
USERIDALIAS mgr1
PURGEOLDEXTRACTS /ogg/dirdat/tt*, USECHECKPOINTS, MINKEEPHOURS 2
```

The following is a sample Manager parameter file on a UNIX system using required and recommended parameters.

Starting Manager

Manager must be running before you start other Oracle GoldenGate processes. You can start Manager from:

- The command line of the operating system. See "[Starting Manager from the Command Shell of the Operating System](#)" for instructions.
- The GGSCI command interface. See "[Starting Manager from GGSCI](#)" for instructions.
- The Services applet on a Windows system if Manager is installed as a service. See the Windows documentation or your system administrator.
- The Cluster Administrator tool if the system is part of a Windows cluster. This is the recommended way to bring the Manager resource online. See the cluster documentation or your system administrator.
- The cluster software of a UNIX or Linux cluster. Refer to the documentation provided by the cluster vendor to determine whether to start Manager from the cluster or by using GGSCI or the command line of the operating system.

- [Starting Manager from the Command Shell of the Operating System](#)
- [Starting Manager from GGSCI](#)

Starting Manager from the Command Shell of the Operating System

To start Manager from the command shell of the operating system, issue the following command.

```
mgr paramfile parameter_file [reportfile report_file]
```

The `reportfile` argument is optional and can be used to store the Manager process report in a location other than the default of the `dirrpt` directory in the Oracle GoldenGate installation location.

Starting Manager from GGSCI

To start Manager from GGSCI, run GGSCI from the Oracle GoldenGate directory, and then issue the following command.

```
START MANAGER
```

Note:

When starting Manager from the command line or GGSCI with User Account Control enabled, you will receive a UAC prompt requesting you to allow or deny the program to run.

Stopping Manager

Manager runs indefinitely or until stopped by a user. In general, Manager should remain running when there are synchronization activities being performed. Manager performs important monitoring and maintenance functions, and processes cannot be started unless Manager is running.

- [Stopping Manager on UNIX and Linux](#)
- [Stopping Manager on Windows](#)

Stopping Manager on UNIX and Linux

On UNIX and Linux (including USS on z/OS), Manager must be stopped by using the `STOP MANAGER` command in GGSCI.

```
STOP MANAGER [!]
```

Where:

! stops Manager without user confirmation.

In a UNIX or Linux cluster, refer to the documentation provided by the cluster vendor to determine whether Manager should be stopped from the cluster or by using GGSCI.

Stopping Manager on Windows

On Windows, you can stop Manager from the Services applet (if Manager is installed as a service). See the Windows documentation or your system administrator.

If Manager is not installed as a service, you can stop it with the `STOP MANAGER` command in GGSCI.

```
STOP MANAGER [!]
```

In a Windows cluster, you must take the Manager resource offline from the Cluster Administrator. If you attempt to stop Manager from the GGSCI interface, the cluster monitor interprets it as a resource failure and attempts to bring the resource online again. Multiple start requests through GGSCI eventually will exceed the start threshold of the Manager cluster resource, and the cluster monitor will mark the Manager resource as failed.

4

Getting Started with the Oracle GoldenGate Process Interfaces

This chapter describes how Oracle GoldenGate users provide instructions to the processes through the GGSCI (Oracle GoldenGate Software Command Interface), batch and shell scripts, and parameter files.

Topics:

- [Using the GGSCI Command-line Interface](#)
- [Controlling Oracle GoldenGate Processes](#)
- [Automating Commands](#)
- [Using Oracle GoldenGate Parameter Files](#)
- [Specifying Object Names in Oracle GoldenGate Input](#)

Using the GGSCI Command-line Interface

You can use GGSCI to issue the complete range of commands that configure, control, and monitor Oracle GoldenGate along with administering secure and non-secured deployments.

GGSCI is the Oracle GoldenGate command-line interface. To start GGSCI, change directories to the Oracle GoldenGate installation directory, and then run the `ggsci` executable file.

Topics:

- [Using Wildcards in Command Arguments](#)
- [Globalization Support for the Command Interface](#)
- [Using Command History](#)
- [Storing and Calling Frequently Used Command Sequences](#)

Using Wildcards in Command Arguments

You can use wildcards with certain Oracle GoldenGate commands to control multiple Extract and Replicat groups as a unit. The wildcard symbol that is supported by Oracle GoldenGate is the asterisk (*). An asterisk represents any number of characters. For example, to start all Extract groups whose names contain the letter X, issue the following command.

```
START EXTRACT *X*
```

Globalization Support for the Command Interface

All command input and related console output are rendered in the default character set of the local operating system. To specify characters that are not compatible with the

character set of the local operating system, use Unicode notation. For example, the following Unicode notation is equivalent to the name of a table that has the Euro symbol as its name:

```
ADD TRANDATA \u20AC1
```

For more information, see [Support for Escape Sequences](#) for more information about using Unicode notation.

**Note:**

Oracle GoldenGate group names are case-insensitive.

Using Command History

The execution of multiple commands is made easier with the following tools:

- Use the `HISTORY` command to display a list of previously executed commands.
- Use the `!` command to execute a previous command again without editing it.
- Use the `FC` command to edit a previous command and then execute it again.

Storing and Calling Frequently Used Command Sequences

You can automate a frequently-used series of commands by using an `OBEY` file and the `OBEY` command. The `OBEY` file takes the character set of the local operating system. To specify a character that is not compatible with that character set, use Unicode notation. See [Support for Escape Sequences](#) for more information about using Unicode notation.

To use OBEY

1. Create and save a text file that contains the commands, one command per line. This is your `OBEY` file. The name can be anything supported by the operating system. You can nest other `OBEY` files within an `OBEY` file.
2. Run GGSCI.
3. (Optional) If using an `OBEY` file that contains nested `OBEY` files, issue the following command. This command enables the use of nested `OBEY` files for the current session of GGSCI and is required whenever using nested `OBEY` files. See *Reference for Oracle GoldenGate* for more information.

```
ALLOWNESTED
```

4. In GGSCI, call the `OBEY` file by using the `OBEY` command.

```
OBEY file_name
```

Where:

file_name is the relative or fully qualified name of the `OBEY` file.

Example 4-1 OBEY command file

```
ADD EXTRACT myext, TRANLOG, BEGIN now
START EXTRACT myext

ADD REPLICAT myrep, EXTTRAIL /ggs/dirdat/aa
START REPLICAT myrep

INFO EXTRACT myext, DETAIL
INFO REPLICAT myrep, DETAIL
```

The following example illustrates an OBEY command file for use with the OBEY command. It creates and starts Extract and Replicat groups and retrieves processing information.

See *Reference for Oracle GoldenGate* for more information about the OBEY command.

Controlling Oracle GoldenGate Processes

The standard way to control Oracle GoldenGate processes is through the GGSCI interface. Typically, the first time that Oracle GoldenGate processes are started in a production setting is during the initial synchronization process (also called *instantiation* process). However, you will need to stop and start the processes at various points as needed to perform maintenance, upgrades, troubleshooting, or other tasks.

These instructions show basic syntax.

Topics:

- [Controlling Manager](#)
- [Controlling Extract and Replicat](#)
- [Deleting Extract and Replicat](#)

Controlling Manager

Manager should not be stopped unless you want to stop replication processing.

To Stop Manager

1. From the Oracle GoldenGate directory, run GGSCI.
2. In GGSCI, issue the following command.

```
{START | STOP [!]} MANAGER
```

Where:

The ! bypasses the prompt that confirms the intent to shut down Manager.

 **Note:**

When starting Manager from the command line or GGSCI with User Account Control enabled, you will receive a UAC prompt requesting you to allow or deny the program to run.

Controlling Extract and Replicat

This section contains basic directions for controlling Extract and Replicat processes. See *Reference for Oracle GoldenGate* for additional command options.

To Start Extract or Replicat

```
START {EXTRACT | REPLICAT} group_name
```

Where:

group_name is the name of the Extract or Replicat group or a wildcard set of groups (for example, * or fin*).

To Stop Extract or Replicat Gracefully

```
STOP {EXTRACT | REPLICAT} group_name
```

Where:

group_name is the name of the Extract or Replicat group or a wildcard set of groups (for example, * or fin*).

To Stop Replicat Forcefully

```
STOP REPLICAT group_name !
```

The current transaction is aborted and the process stops immediately. You cannot stop Extract forcefully.

To Kill a Process that STOP Cannot Stop

```
KILL {EXTRACT | REPLICAT} group_name
```

Killing a process does not shut it down gracefully, and checkpoint information can be lost.

To Control Multiple Processes at Once

```
command ER wildcard specification
```

Where:

- *command* is: KILL, START, or STOP
- *wildcard specification* is a wildcard specification for the names of the process groups that you want to affect with the command. The command affects every Extract and Replicat group that satisfies the wildcard. Oracle GoldenGate supports up to 100,000 wildcard entries.

Deleting Extract and Replicat

This section contains basic directions for deleting Extract and Replicat processes. See *Reference for Oracle GoldenGate* for additional command options.

To Delete an Extract Group

1. Run GGSCI.

2. Issue the `DBLOGIN` command as the Extract database user (or a user with the same privileges). You can use either of the following commands, depending on whether a local credential store exists.

```
DBLOGIN [SOURCEDB dsn] {USERID user, PASSWORD password [encryption_options]
| USERIDALIAS alias [DOMAIN domain]}
```

3. Stop the Extract process.

```
STOP EXTRACT group_name
```

4. Issue the following command.

```
DELETE EXTRACT group_name
```

5. (Oracle) Unregister the Extract group from the database.

```
UNREGISTER EXTRACT group_name,database_name
```

To Delete a Replicat Group

1. Stop the Replicat process.

```
STOP REPLICAT group_name
```

2. Issue one of the following commands from GGSCI to log into the database.

```
DBLOGIN [SOURCEDB dsn] {USERID user, PASSWORD password [encryption_options]
| USERIDALIAS alias [DOMAIN domain]}
```

Where:

- `SOURCEDB dsn` supplies the data source name, if required as part of the connection information.
 - `USERID user, PASSWORD password` specifies an explicit database login credential.
 - `USERIDALIAS alias [DOMAIN domain]` specifies an alias and optional domain of a credential that is stored in a local credential store.
 - `encryption_options` is one of the options that encrypt the password.
3. Issue the following command to delete the group.

```
DELETE REPLICAT group_name
```

Deleting a Replicat group preserves the checkpoints in the checkpoint table (if being used). Deleting a process group also preserves the parameter file. You can create the same group again, using the same parameter file, or you can delete the parameter file to remove the group's configuration permanently.

Automating Commands

Oracle GoldenGate supports the issuing of commands through scripts or jobs. This section describes these options for UNIX- or Linux-based platforms and the IBMi platform.

On a UNIX or Linux system, or within a runtime environment that supports UNIX or Linux applications, you can issue Oracle GoldenGate commands from a script such as a startup script, shutdown script, or failover script by running GGSCI and calling an input file. The script file must be encoded in the operating system character set. Unicode notation can be used for characters that are not supported by the operating

system character set. Before creating a script, see [Globalization Support for the Command Interface](#).

To Input a Script

Use the following syntax from the command line of the operating system.

```
ggsci < input_file
```

Where:

- The angle bracket (<) character pipes the file into the GGSCI program.
- *input_file* is a text file, known as an OBEY file, containing the commands that you want to issue, in the order they are to be issued.

Note:

To stop the Manager process from a batch file, make certain to add the ! argument to the end of the STOP MANAGER command. Otherwise, GGSCI issues a prompt that requires a response and causes the process to enter into a loop. See [Stopping Manager](#) for more information about stopping Manager.

- [Issuing Commands Through the IBM i CLI](#)

Issuing Commands Through the IBM i CLI

Oracle GoldenGate for IBM DB2 for i includes a set of native IBM i commands that enables the operation of the most common Oracle GoldenGate programs from the IBM i command-line interface (CLI). Because these commands are native, they do not need to be run from a PASE environment. With this support, it is possible to issue commands interactively or by using the typical job submission tools such as SBMJOB to operate Oracle GoldenGate non-interactively.

The commands are as follows and correspond to the Oracle GoldenGate programs of the same name. They reside in the Oracle GoldenGate installation library.

DEFGEN

EXTRACT

GGSCI

KEYGEN

LOGDUMP

MGR

REPLICAT

For more information about these commands, see [Reference for Oracle GoldenGate for Windows and UNIX](#).

Using Oracle GoldenGate Parameter Files

Most Oracle GoldenGate functionality is controlled by means of parameters specified in parameter files. A parameter file is a plain text file that is read by an associated Oracle GoldenGate process. Oracle GoldenGate uses two types of parameter files: a GLOBALS file and runtime parameter files.

Topics:

- [Globalization Support for Parameter Files](#)
- [Working with the GLOBALS File](#)
- [Working with Runtime Parameters](#)
- [Creating a Parameter File](#)
- [Validating a Parameter File](#)
- [Viewing a Parameter File](#)
- [Changing a Parameter File](#)
- [Simplifying the Creation of Parameter Files](#)
- [Getting Information about Oracle GoldenGate Parameters](#)

Globalization Support for Parameter Files

Oracle GoldenGate creates parameter files in the default character set of the local operating system. In the event that the local platform does not support a required character set as the default in the operating system, you can use the `CHARSET` parameter either globally or per-process to specify a character set for parameter files.

To avoid issues caused by character-set incompatibilities, create or edit a parameter file on the server where the associated process will be running. Avoid creating it on one system (such as your Windows laptop) and then transferring the file to the UNIX server where Oracle GoldenGate is installed and where the operating system character set is different. Oracle GoldenGate provides some tools to help with character set incompatibilities if you must create the parameter file on a different system:

- You can use the `CHARSET` parameter to specify a compatible character set for the parameter file. This parameter must be placed on the first line of the parameter file and allows you to write the file in the specified character set. After the file is transferred to the other system, do not edit the file on that system.
- You can use Unicode notation to substitute for characters that are not compatible with the character set of the operating system where the file will be used. See [Support for Escape Sequences](#) for more information about Unicode notation.

See *Reference for Oracle GoldenGate* for more information about the `CHARSET` parameter.

Working with the GLOBALS File

The GLOBALS file stores parameters that relate to the Oracle GoldenGate instance as a whole. This is in contrast to runtime parameters, which are coupled with a specific process such as Extract. The parameters in the GLOBALS file apply to all processes in

the Oracle GoldenGate instance, but can be overridden by specific process parameters. A `GLOBALS` parameter file may or may not be required for your Oracle GoldenGate environment.

When used, a `GLOBALS` file must exist before starting any Oracle GoldenGate processes, including GGSCI. The GGSCI program reads the `GLOBALS` file and passes the parameters to processes that need them.

To Create a GLOBALS File

1. From the Oracle GoldenGate installation location, run GGSCI and enter the following command, or open a file in a text editor.

```
EDIT PARAMS ./GLOBALS
```

Note:

The `./` portion of this command must be used, because the `GLOBALS` file must reside at the root of the Oracle GoldenGate installation file.

2. In the file, enter the `GLOBALS` parameters, one per line.
3. Save the file. If you used a text editor, save the file as `GLOBALS` (uppercase, without a file extension) at the root of the Oracle GoldenGate installation directory. If you created the file correctly in GGSCI, the file is saved that way automatically. Do not move this file.
4. Exit GGSCI. You must start from a new GGSCI session before issuing commands or starting processes that reference the `GLOBALS` file.

Working with Runtime Parameters

Runtime parameters give you control over the various aspects of Oracle GoldenGate synchronization, such as:

- Data selection, mapping, transformation, and replication
- DDL and sequence selection, mapping, and replication (where supported)
- Error resolution
- Logging
- Status and error reporting
- System resource usage
- Startup and runtime behavior

There can be only one active parameter file for the Manager process or an Extract or Replicat group; however, you can use parameters in other files by using the `OBEY` parameter. See [Simplifying the Creation of Parameter Files](#) for more information about simplifying the use of parameter files.

There are two types of parameters: global (not to be confused with `GLOBALS` parameters) and object-specific:

- Global parameters apply to all database objects that are specified in a parameter file. Some global parameters affect process behavior, while others affect such

things as memory utilization and so forth. `USERIDALIAS` in [Example 4-2](#) and [Example 4-3](#) is an example of a global parameter. In most cases, a global parameter can appear anywhere in the file before the parameters that specify database objects, such as the `TABLE` and `MAP` statements in [Example 4-2](#) and [Example 4-3](#). A global parameter should be listed only once in the file. When listed more than once, only the *last* instance is active, and all other instances are ignored.

- Object-specific parameters enable you to apply different processing rules for different sets of database objects. `GETINSERTS` and `IGNOREINSERTS` in [Example 4-3](#) are examples of object-specific parameters. Each precedes a `MAP` statement that specifies the objects to be affected. Object-specific parameters take effect in the order that each one is listed in the file.

[Example 4-2](#) and [Example 4-3](#) are examples of basic parameter files for Extract and Replicat. Comments are preceded by double hyphens.

The preceding example reflects a case-insensitive Oracle database, where the object names are specified in the `TABLE` statements in capitals. For a case-insensitive Oracle database, it makes no difference how the names are entered in the parameter file (upper, lower, mixed case). For other databases, the case of the object names may matter. See [Specifying Object Names in Oracle GoldenGate Input](#) for more information about specifying object names.

Note the use of single and double quote marks in the Replicat example in [Example 4-3](#). For databases that require quote marks to enforce case-sensitive object names, such as Oracle, you must enclose case-sensitive object names within double quotes in the parameter file as well. For other case-sensitive databases, specify the names as they are stored in the database. For more information about specifying names and literals, see [Specifying Object Names in Oracle GoldenGate Input](#).

Example 4-2 Sample Extract Parameter File

```
-- Extract group name
EXTRACT capt
-- Extract database user login, with alias to credentials in the credential store.
USERIDALIAS ogg1
-- Remote host to where captured data is sent in encrypted format:
RMTHOSTOPTIONS sysb, MGRPORT 7809, ENCRYPT AES192 KEYNAME mykey
-- Encryption specification for trail data
ENCRYPTTRAIL AES192
-- Remote trail on the remote host
RMTRAIL /ggs/dirdat/aa
-- TABLE statements that identify data to capture.
TABLE FIN.*;
TABLE SALES.*;
```

Example 4-3 Sample Replicat Parameter File

```
-- Replicat group name
REPLICAT deliv
-- Replicat database user login, with alias to credentials in the credential store
USERIDALIAS ogg2
-- Error handling rules
REPERROR DEFAULT, ABEND
-- Ignore INSERT operations
IGNOREINSERTS
-- MAP statement to map source objects to target objects and
-- specify column mapping
MAP "fin"."accTAB", TARGET "fin"."accTAB",
```

```
COLMAP ("Account" = "Acct",
"Balance" = "Bal",
"Branch" = "Branch");
-- Get INSERT operations
GETINSERTS
-- MAP statement to map source objects to target objects and
-- filter to apply only the 'NY' branch data.
MAP "fin"."teller", TARGET "fin"."tellTAB",
WHERE ("Branch" = 'NY');
```

Creating a Parameter File

Oracle recommends using GGSCI when writing the parameter file in the character set of the operating system, but if using the `CHARSET` parameter and writing the file in a different character set, use a text editor instead of GGSCI.

Topics:

- [Creating a Parameter File in GGSCI](#)
- [Creating a Parameter File with a Text Editor](#)

Creating a Parameter File in GGSCI

To create a parameter file, use the `EDIT PARAMS` command within the GGSCI user interface or use a text editor directly. When you use GGSCI, you are using a standard text editor, but your parameter file is saved automatically with the correct file name and in the correct directory.

When you create a parameter file with `EDIT PARAMS` in GGSCI, it is saved to the `dirprm` sub-directory of the Oracle GoldenGate directory. You can create a parameter file in a directory other than `dirprm`, but you also must specify the full path name with the `PARAMS` option of the `ADD EXTRACT` or `ADD REPLICAT` command when you create your process groups. Once paired with an Extract or Replicat group, a parameter file must remain in its original location for Oracle GoldenGate to operate properly once processing has started.

The `EDIT PARAMS` command launches the following text editors within the GGSCI interface:

- Notepad on Microsoft Windows systems
- The vi editor on UNIX and Linux systems. DB2 for i only supports vi when connected with SSH or xterm. For more information, see [Creating a Parameter File with a Text Editor](#).

Note:

You can change the default editor through the GGSCI interface by using the `SET EDITOR` command. See *Reference for Oracle GoldenGate*.

1. From the directory where Oracle GoldenGate is installed, run GGSCI.
2. In GGSCI, issue the following command to open the default text editor.

```
EDIT PARAMS group_name
```

Where:

group_name is either `mgr` (for the Manager process) or the name of the Extract or Replicat group for which the file is being created. The name of an Extract or Replicat parameter file must match that of the process group.

The following creates or edits the parameter file for an Extract group named `extora`.

```
EDIT PARAMS extora
```

The following creates or edits the parameter file for the Manager process.

```
EDIT PARAMS MGR
```

3. Using the editing functions of the text editor, enter as many comment lines as you want to describe this file, making certain that each comment line is preceded with two hyphens (--).
4. On non-commented lines, enter the Oracle GoldenGate parameters, starting a new line for each parameter statement.

Oracle GoldenGate parameters have the following syntax:

```
PARAMETER_NAME argument [, option] [&]
```

Where:

- *PARAMETER_NAME* is the name of the parameter.
- *argument* is a required argument for the parameter. Some parameters take arguments, but others do not. Commas between arguments are optional.

```
EXTRACT myext
USERIDALIAS ogg1
RMTHOSTOPTIONS sysb, MGRPORT 8040, ENCRYPT AES192 KEYNAME mykey
ENCRYPTTRAIL AES 192
RMTRAIL /home/ggs/dirdat/c1, PURGE
CUSEREXIT userexit.dll MyUserExit, INCLUDEUPDATEBEFORES, &
  PARAMS "init.properties"
TABLE myschema.mytable;
```

- *[, option]* is an optional argument.
- *[&]* is required at the end of each line in a multi-line parameter statement, as in the `CUSEREXIT` parameter statement in the previous example. The exceptions are the following, which can accept, but do not require, the ampersand because they terminate with a semicolon:

- MAP
- TABLE
- SEQUENCE
- FILE
- QUERY

 **Note:**

The `RMTHOST` and `RMTHOSTOPTIONS` parameters can be specified together; the `RMTHOST` parameter is *not* required for `RMTHOSTOPTIONS` if the dynamic IP assignment is properly configured. When `RMTHOSTOPTIONS` is used, the `MGRPORT` option is ignored.

5. Save and close the file.

Creating a Parameter File with a Text Editor

You can create a parameter file outside GGSCI by using a text editor, but make certain to:

- Save the parameter file with the name of the Extract or Replicat group that owns it, or save it with the name `mgr` if the Manager process owns it. Use the `.prm` file extension. For example: `extfin.prm` and `mgr.prm`.
- Save the parameter file in the `dirprm` directory of the Oracle GoldenGate installation directory.
- For DB2 for i systems, you can edit parameter files from a 5250 terminal using SEU or EDTF. If you use SEU, you must copy the file using the `CPYTOSTMF` command, specify an encoding of CCSID 1208, and line endings of *LF. If editing with EDTF from F15 (services) ensure that you change the CCSID of the file to 1208 and the EOL option to *LF.

Alternatively, you can use the `Rfile` command from the IBM Portable Application Solutions Environment for i.

Validating a Parameter File

The `checkprm` validation native command is run from the command line and gives an assessment of the specified parameter file, with a configurable application and running environment. It can provide either a simple `PASS/FAIL` or with optional details about how the values of each parameter are stored and interpreted.

The input to `checkprm` is case insensitive. If a value string contains spaces, it does not need to be quoted because `checkprm` can recognize meaningful values. If no mode is specified to `checkprm`, then all parameters applicable to any mode of the component will be accepted.

The output of `checkprm` is assembled with four possible sections:

- help messages
- pre-validation error
- validation result
- parameter details

A pre-validation error is typically an error that prevents a normal parameter validation from executing, such as missing options or an inaccessible parameter file. If an option value is specified incorrectly, a list of possible inputs for that option is provided. If the result is `FAIL`, each error is in the final result message. If the result is `PASS`, a message that some of the parameters are subject to further runtime validation. The parameter

detailed output contains the validation context, the values read from GLOBALS (if it is present), and the specified parameters. The parameter and options are printed with proper indentation to illustrate these relationships.

Table 4-1 describes all of the arguments that you can use with the `checkprm` commands. When you use `checkprm` and do not use any of these arguments, then `checkprm` attempts to automatically detect Extract or Replicat and the platform and database of the Oracle GoldenGate installation.

Table 4-1 checkprm Arguments

Argument	Purpose & Behavior
None	Displays usage information
-v	Displays banner. Cannot be combined with other options.
? help	Displays detailed usage information, include all possible values of each option. Cannot be combine with other options.
<i>parameter_file</i>	Specifies the name of the parameter file, has to be the first argument if a validation is requested. You must specify the absolute path to the parameter file. For example, <code>CHECKPRM ./dirprm/myext.prm</code> .
-COMPONENT -C	Specifies the running component (application) that this parameter file is validated for. This option can be omitted for Extract or Replicat because automatic detection is attempted. Valid values include: CACHEFILEDUMP COBGEN CONVCHK CONVPRM DDLCOB DEFGEN EMSCLNT EXTRACT GGCMD GGSCI KEYGEN LOGDUMP MGR OGGERR REPLICAT RETRACE REVERSE SERVER GLOBALS There is no default for this option.
-MODE -M	Specifies the mode of the running application if applicable. This option is optional, only applicable to Extract or Replicat. If no mode is specified, the validation is performed for all Extract or Replicat modes. Valid input of this option includes: <ul style="list-style-type: none"> • Classic Extract • Integrated Extract • Initial Load Extract • Remote Task Extract • Data Pump Extract • Passive Extract • Classic Replicat • Coordinated Replicat • Integrated Replicat • Parallel Integrated Replicat • Parallel Nonintegrated Replicat • Special Run Replicat • Remote Task When key in the value for this option, the application name is optional, as long as it matches the value of component. For example, "Data Pump Extract" is equivalent to "Data Pump" if the component is Extract. However, it is invalid if the component is Replicat.

Table 4-1 (Cont.) checkprm Arguments

Argument	Purpose & Behavior
-PLATFORM -P	<p>Specifies the platform the application is supposed to run on. The default value is the platform that this checkprm executable is running on.</p> <p>The possible values are:</p> <p>AIX HP-OSS HPUX-IT HPUX-PA Linux OS400 ZOS Solaris SPARC Solaris x86 Windows x64 All</p>
-DATABASE -D	<p>Specifies the database the application is built against. The default value is the database for your Oracle GoldenGate installation.</p> <p>The database options are (case insensitive):</p> <p>Generic Oracle 8 Oracle 9i Oracle 10g Oracle 11g Oracle 12c Sybase DB2LUW 9.5 DB2LUW 9.7 DB2LUW 10.5 DB2LUW 10.1 DB2 Remote Teradata Timesten Timesten 7 Timesten 11.2.1 MySQL Ctree8 Ctree9 DB2 for i DB2 for i Remote MS SQL MS SQL CDC Informix Informix1150 Informix1170 Informix1210 Ingres SQL/MX DB2 z/OS PostgreSQL</p>
-VERBOSE -V	<p>Directs checkprm to print out detailed parameter information, to demonstrate how the values are read and interpreted.</p> <p>It must be the last option specified in a validation.</p>

Following are some use examples:

```
checkprm ?
checkprm ./dirprm/ext1.prm -C extract -m data pump -p Linux -v
checkprm ./dirprm/ext1.prm -m integrated
checkprm ./dirprm/repl.prm -m integrated
checkprm ./dirprm/mgr.prm -C mgr -v
checkprm GLOBALS -c GLOBALS
```

Verifying Using CHECKPARAMS Parameter

An alternative to using the recommended `checkprm` utility, is to check the syntax of parameters in an Extract or Replicat parameter file for accuracy using the `CHECKPARAMS` parameter. This process can be used with Extract or Replicat.

To Verify Parameter Syntax

1. Include the `CHECKPARAMS` parameter in the parameter file.
2. Start the associated process by issuing the `START EXTRACT` or `START REPLICAT` command in GGSCI.

```
START {EXTRACT | REPLICAT} group_name
```

The process audits the syntax, writes the results to the report file or the screen, and then stops.

3. Do either of the following:
 - If the syntax is correct, remove the `CHECKPARAMS` parameter before starting the process to process data.
 - If the syntax is wrong, correct it based on the findings in the report. You can run another test to verify the changes, if desired. Remove `CHECKPARAMS` before starting the process to process data.

For more information about the report file, see [Monitoring Oracle GoldenGate Processing](#).

For more information about `CHECKPARAMS`, see *Reference for Oracle GoldenGate*.

Viewing a Parameter File

You can view a parameter file directly from the command shell of the operating system, or you can view it from the GGSCI user interface. To view the file from GGSCI, use the `VIEW PARAMS` command.

```
VIEW PARAMS group_name
```

Where:

`group_name` is either `mgr` (for Manager) or the name of the Extract or Replicat group that is associated with the parameter file.

▲ Caution:

Do not use `VIEW PARAMS` to view an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). The contents may become corrupted. View the parameter file from outside GGSCI.

If the parameter file was created in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory, specify the full path name as shown in the following example.

```
VIEW PARAMS c:\lpparms\replp.prm
```

Changing a Parameter File

An Oracle GoldenGate process must be stopped before changing its parameter file, and then started again after saving the parameter file. Changing parameter settings while a process is running can have unexpected results, especially if you are adding tables or changing mapping or filtering rules.

▲ Caution:

Do not use the `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). The contents may become corrupted. View the parameter file from outside GGSCI.

To Change Parameters:

1. Stop the process by issuing the following command in GGSCI. To stop Manager in a Windows cluster, use the Cluster Administrator.

```
STOP {EXTRACT | REPLICAT | MANAGER} group_name
```

2. Open the parameter file by using a text editor or the `EDIT PARAMS` command in GGSCI.

```
EDIT PARAMS mgr
```

3. Make the edits, and then save the file.
4. Start the process by issuing the following command in GGSCI. Use the Cluster Administrator if starting Manager in a Windows cluster.

```
START {EXTRACT | REPLICAT | MANAGER} group_name
```

Simplifying the Creation of Parameter Files

You can reduce the number of times that a parameter must be specified by using the following time-saving tools.

Topics:

- [Using Wildcards](#)
- [Using OBEY](#)
- [Using Macros](#)
- [Using Parameter Substitution](#)

Using Wildcards

For parameters that accept object names, you can use asterisk (*) and question mark (?) wildcards. The use of wildcards reduces the work of specifying numerous object names or all objects within a given schema. For more information about using wildcards, see [Using Wildcards in Database Object Names](#).

Using OBEY

You can create a library of text files that contain frequently used parameter settings, and then you can call any of those files from the active parameter file by means of the OBEY parameter. The syntax for OBEY is:

```
OBEY file_name
```

Where:

file_name is the relative or full path name of the file.

Upon encountering an OBEY parameter in the active parameter file, Oracle GoldenGate processes the parameters from the referenced file and then returns to the active file to process any remaining parameters. OBEY is not supported for the GLOBALS parameter file.

If using the CHARSET parameter in a parameter file that includes an OBEY parameter, the referenced parameter file does not inherit the CHARSET character set. The CHARSET character set is used to read wildcarded object names in the referenced file, but you must use an escape sequence (\uX) for all other multibyte specifications in the referenced file.

See *Reference for Oracle GoldenGate* for more information about OBEY.

See *Reference for Oracle GoldenGate* for more information about CHARSET.

Using Macros

You can use macros to automate multiple uses of a parameter statement. See [Using Oracle GoldenGate Macros to Simplify and Automate Work](#).

Using Parameter Substitution

You can use parameter substitution to assign values to Oracle GoldenGate parameters automatically at run time, instead of assigning static values when you create the parameter file. That way, if values change from run to run, you can avoid having to edit the parameter file or maintain multiple files with different settings. You can simply export the required value at runtime. Parameter substitution can be used for any Oracle GoldenGate process.

To Use Parameter Substitution

1. For each parameter for which substitution is to occur, declare a runtime parameter instead of a value, and precede the runtime parameter name with a question mark (?) as shown in the following example.

```
SOURCEISFILE
EXTFILE ?EXTFILE
MAP scott?TABNAME, TARGET tiger ACCOUNT_TARG;
```

2. Before starting the Oracle GoldenGate process, use the shell of the operating system to pass the runtime values by means of an environment variable, as shown in [Example 4-4](#) and [Example 4-5](#).

Example 4-4 Parameter substitution on Windows

```
C:\GGS> set EXTFILE=C:\ggs\extfile
C:\GGS> set TABNAME=PROD.ACCOUNTS
C:\GGS> replicat paramfile c:\ggs\dirprm\parmfl
```

Example 4-5 Parameter substitution on UNIX (Korn shell)

```
$ EXTFILE=/ggs/extfile
$ export EXTFILE
$ TABNAME=PROD.ACCOUNTS
$ export TABNAME
$ replicat paramfile ggs/dirprm/parmfl
```

UNIX is case-sensitive, so the parameter declaration in the parameter file must be the same case as the shell variable assignments.

Getting Information about Oracle GoldenGate Parameters

You can use the `INFO PARAM` command to view a parameter's definition information from GGSCI. The name provided in the command line can be a parameter, or an option, but it must be a full name that is part of the names concatenated together using a period (.) as the delimiter. For example:

```
INFO PARAM RMTHOST
RMTHOST.STREAMING
INFO PARAM RMTHOST.STREAMING
```

Using the `GETPARAMINFO`, you can query the runtime parameter values of a running instance, including Extract, Replicat, and Manager. This command is similar to using `checkprm -v`, see [Validating a Parameter File](#). The default behavior is to display all that has ever been queried by the application, parameters and their current values. If a particular parameter name is specified, then the output is filtered by that name. Optionally, the output can be redirect to a file specified by the `-FILE` option. For example:

```
SEND extlpm GETPARAMINFO
```

For more information about these and all Oracle GoldenGate parameters including exact syntax, see the *Reference for Oracle GoldenGate*.

Specifying Object Names in Oracle GoldenGate Input

The following rules apply when specifying object names in parameter files (such as in `TABLE` and `MAP` statements), column-conversion functions, commands, and in other input.

Topics:

- [Specifying Filesystem Path Names in Parameter Files on Windows Systems](#)
- [Supported Database Object Names](#)
- [Specifying Names that Contain Slashes](#)
- [Qualifying Database Object Names](#)
- [Specifying Case-Sensitive Database Object Names](#)
- [Using Wildcards in Database Object Names](#)
- [Differentiating Case-Sensitive Column Names from Literals](#)

Specifying Filesystem Path Names in Parameter Files on Windows Systems

On Windows systems, if the name of any directory in a filesystem path name begins with a number, the path must be specified with forward slashes, not backward slashes, when listing that path in Oracle GoldenGate input, such as parameter files or commands. This requirement prevents Oracle GoldenGate from interpreting the name as an octal escape sequence. For example, the following paths contain a directory named `\2014` that will be interpreted as the octal sequence `\201`:

```
C:\ogg\2014\install\dir\aa
C:\ogg\install\2014\dir\aa
```

The preceding path can be used with forward slashes as follows:

```
C:/ogg/2014/install/dir/aa
C:/ogg/install/2014/dir/aa
```

For more information, see [Support for Escape Sequences](#).

Supported Database Object Names

Object names in parameter files, command, and other input can be any length and in any supported character set. For supported character sets, see [Supported Character Sets](#).

Oracle GoldenGate supports most characters in object and column names. Specify object names in double quote marks if they contain special characters such as white spaces or symbols.

The following lists of supported and non-supported characters covers all databases supported by Oracle GoldenGate; a given database platform may or may not support all listed characters.

Topics:

- [Supported Special Characters](#)
- [Non-supported Special Characters](#)

Supported Special Characters

Oracle GoldenGate supports all characters that are supported by the database, including the following special characters. Object names that contain these special characters must be enclosed within double quotes in parameter files.

Character	Description
/	Forward slash (See Specifying Names that Contain Slashes)
*	Asterisk (Must be escaped by a backward slash when used in parameter file, as in: *)
?	Question mark (Must be escaped by a backward slash when used in parameter file, as in: \?)
@	At symbol (Supported, but is often used as a resource locator by databases. May cause problems in object names)
#	Pound symbol
\$	Dollar symbol
%	Percent symbol (Must be %% when used in parameter file)
^	Caret symbol
()	Open and close parentheses
_	Underscore
-	Dash
<space>	Space

Non-supported Special Characters

The following characters are not supported in object names and non-key column names.

Character	Description
\	Backward slash (Must be \\ when used in parameter file)
{ }	Begin and end curly brackets (braces)
[]	Begin and end brackets
=	Equal symbol
+	Plus sign
!	Exclamation point
~	Tilde
	Pipe
&	Ampersand
:	Colon
;	Semi-colon
,	Comma

Character	Description
'	Single quotes
"	Double quotes
´	Accent mark (Diacritical mark)
.	Period
<	Less-than symbol (or beginning angle bracket)
>	Greater-than symbol (or ending angle bracket)

Specifying Names that Contain Slashes

If a table name contains a forward-slash character (/) in any part of its name, that name component must be enclosed within double quotes unless the object name is from an IBM i platform. The following are some examples:

```
"c/d"  
"/a".b  
a."b/"
```

If the name contains a forward slash that is not enclosed within double quotes, Oracle GoldenGate treats it as a name that originated on the IBM i platform (from a DB2 for i database). The forward slash in the name is interpreted as a separator character.

Qualifying Database Object Names

Object names must be fully qualified in the parameter file. This means that every name specification must be qualified, not only those supplied as input to Oracle GoldenGate parameter syntax, but also names in a SQL procedure or query that is supplied as `SQLEXEC` input, names in user exit input, and all other input supplied in the parameter file.

Oracle GoldenGate supports two-part and three-part object names, as appropriate for the database.

Topics:

- [Two-part Names](#)
- [Three-part Names](#)
- [Applying Data from Multiple Containers or Catalogs](#)
- [Specifying a Default Container or Catalog](#)

Two-part Names

Most databases require only two-part names to be specified, in the following format:

```
owner.object
```

For example: `HR.EMP`

Where:

owner is a schema or database, depending on how the database defines a logical namespace that contains database objects. *object* is a table or other supported database object.

The databases for which Oracle GoldenGate supports two-part names are as follows, shown with their appropriate two-part naming convention:

- DB2 for i: *schema.object* and *library/file(member)*
- DB2 LUW: *schema.object*
- DB2 on z/OS: *schema.object*
- MySQL: *database.object*
- Oracle Database (non-CDB databases): *schema.object*
- SQL Server: *schema.object*
- Teradata: *database.object*

Three-part Names

Oracle GoldenGate supports three-part names for the following databases:

- Oracle container databases (CDB)

Three-part names are required to capture from a source Oracle container database because one Extract group can capture from more than one container. Thus, the name of the container, as well as the schema, must be specified for each object or objects in an Extract `TABLE` statement.

Specify a three-part Oracle CDB name as follows:

container.schema.object

For example: `PDB1.HR.EMP`

Applying Data from Multiple Containers or Catalogs

To apply data captured from multiple source containers or catalogs to a target Oracle container database, both three- and two-part names are required. In the `MAP` portion of the `MAP` statement, each source object must be associated with a container or catalog, just as it was in the `TABLE` statement. This enables you (and Replicat) to properly map data from multiple source containers or catalogs to the appropriate target objects. In the `TARGET` portion of the `MAP` statement, however, only two-part names are required. This is because Replicat can connect to only one target container or catalog at a time, and *schema.owner* is a sufficient qualifier. Multiple Replicat groups are required to support multiple target containers or catalogs. Specify the target container or catalog with the `TARGETDB` parameter.

Specifying a Default Container or Catalog

You can use the `SOURCECATALOG` parameter to specify a default catalog for any subsequent `TABLE`, `MAP`, (or Oracle `SEQUENCE`) specifications in the parameter file. The following example shows the use of `SOURCECATALOG` to specify the default Oracle PDB named `pdb2` for `schema2` and `schema3` objects, and the default PDB named `pdb3` for `schema4` objects. The objects in `pdb1` are specified with a fully qualified three-part name, which does not require a default catalog to be specified.

```
TABLE pdb1.schema1.table*;
SOURCECATALOG pdb2
TABLE schema2.table*;
TABLE schema3.table*;
SOURCECATALOG pdb3
TABLE schema4.table*;
```

Specifying Case-Sensitive Database Object Names

Oracle GoldenGate supports case-sensitive names. Follow these rules when specifying case-sensitive objects.

- Specify object names from a case-sensitive database in the same case that is used to store them in the host database. Keep in mind that, in some database types, different levels of the database can have different case-sensitivity, such as case-sensitive schema but case-insensitive table. If the database requires quotes to enforce case-sensitivity, put quotes around each object that is case-sensitive in the qualified name.

Correct: TABLE "Sales"."ACCOUNT"

Incorrect: TABLE "Sales.ACCOUNT"

- Oracle GoldenGate converts case-insensitive names to the case in which they are stored when required for mapping purposes.

Table 4-2 provides an overview of the support for case-sensitivity in object names, per supported database. Refer to the database documentation for details on this type of support.

Table 4-2 Case Sensitivity of Object Names Per Database

Database	Requires quotes to enforce case-sensitivity?	Unquoted object name	Quoted object name
DB2	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in upper case	Case-sensitive, stores in mixed case
MySQL (Case-sensitive database)	No <ul style="list-style-type: none"> Always case-sensitive, stores in mixed case The names of columns, triggers, and procedures are case-insensitive 	No effect	No effect
Oracle Database	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in upper case	Case-sensitive, stores in mixed case
SQL Server (Database created as case-sensitive)	No <ul style="list-style-type: none"> Always case-sensitive, stores in mixed case 	No effect	No effect

Table 4-2 (Cont.) Case Sensitivity of Object Names Per Database

Database	Requires quotes to enforce case-sensitivity?	Unquoted object name	Quoted object name
SQL Server (Database created as case-insensitive)	No Always case-insensitive, stores in mixed case	No effect	No effect
Teradata	No Always case-insensitive, stores in mixed case	No effect	No effect

 **Note:**

For all supported databases, passwords are always treated as case-sensitive regardless of whether the associated object name is quoted or unquoted.

Using Wildcards in Database Object Names

You can use wildcards for any part of a fully qualified object name, if supported for the specific database. These name parts can be the following: the container, database, or catalog name, the owner (schema or database name), and table or sequence name. For specifics on how object names and wildcards are supported, see the Oracle GoldenGate installation and configuration guide for that database.

Where appropriate, Oracle GoldenGate parameters permit the use of two wildcard types to specify multiple objects in one statement:

- A question mark (?) replaces one character. For example in a schema that contains tables named `TABn`, where n is from 0 to 9, a wildcard specification of `HQ.TAB?` returns `HQ.TAB0`, `HQ.TAB1`, `HQ.TAB2`, and so on, up to `HQ.TAB9`, but no others. This wildcard is not supported for the DB2 LUW database nor for DEFGEN. This wildcard can only be used to specify source objects in a `TABLE` or `MAP` parameter. It cannot be used to specify target objects in the `TARGET` clause of `TABLE` or `MAP`.
- An asterisk (*) represents any number of characters (including zero sequence). For example, the specification of `HQ.T*` could return such objects as `HQ.TOTAL`, `HQ.T123`, and `HQ.T`. This wildcard is valid for all database types throughout all Oracle GoldenGate commands and parameters where a wildcard is allowed.
- In `TABLE` and `MAP` statements, you can combine the asterisk and question-mark wildcard characters in source object names only.

Topics:

- [Rules for Using Wildcards for Source Objects](#)
- [Rules for Using Wildcards for Target Objects](#)
- [Fallback Name Mapping](#)

- [Wildcard Mapping from Pre-11.2.1 Trail Version](#)
- [Asterisks or Question Marks as Literals in Object Names](#)
- [How Wildcards are Resolved](#)
- [Excluding Objects from a Wildcard Specification](#)

Rules for Using Wildcards for Source Objects

For source objects, you can use the asterisk alone or with a partial name. For example, the following source specifications are valid:

- `TABLE HQ.*;`
- `TABLE PDB*.HQ.*;`
- `MAP HQ.T_*;`
- `MAP HQ.T_*, TARGET HQ.*;`

The `TABLE`, `MAP` and `SEQUENCE` parameters take the case-sensitivity and locale of the database into account for wildcard resolution. For databases that are created as case-sensitive or case-insensitive, the wildcard matches the exact name and case. For example, if the database is case-sensitive, `SCHEMA.TABLE` is matched to `SCHEMA.TABLE`, `Schema.Table` is matched to `Schema.Table`, and so forth. If the database is case-insensitive, the matching is not case-sensitive.

For databases that can have both case-sensitive and case-insensitive object names in the same database instance, with the use of quote marks to enforce case-sensitivity, the wildcarding works differently. When used alone for a source name in a `TABLE` statement, an asterisk wildcard matches any character, whether or not the asterisk is within quotes. The following statements produce the same results:

```
TABLE hr.*;  
TABLE hr."*";
```

Similarly, a question mark wildcard used alone matches any single character, whether or not it is within quotes. The following produce the same results:

```
TABLE hr.?  
TABLE hr."?";
```

If a question mark or asterisk wildcard is used with other characters, case-sensitivity is applied to the non-wildcard characters, but the wildcard matches both case-sensitive and case-insensitive names.

- The following `TABLE` statements capture any table name that begins with lower-case `abc`. The quoted name case is preserved and a case-sensitive match is applied. It captures table names that include `"abcA"` and `"abca"` because the wildcard matches both case-sensitive and case-insensitive characters.

```
TABLE hr."abc*";  
TABLE hr."abc?";
```

- The following `TABLE` statements capture any table name that begins with upper-case `ABC`, because the partial name is case-insensitive (no quotes) and is stored in upper case by this database. However, because the wildcard matches both case-sensitive and case-insensitive characters, this example captures table names that include `ABCA` and `"ABCa"`.

```
TABLE hr.abc*;  
TABLE hr.abc?;
```

Rules for Using Wildcards for Target Objects

When using wildcards in the `TARGET` clause of a `MAP` statement, the target objects must exist in the target database. (The exception is when DDL replication is being used, which allows new schemas and their objects to be replicated as they are created.)

For target objects, only an asterisk can be used. If an asterisk wildcard is used with a partial name, Replicat replaces the wildcard with the entire name of the corresponding source object. Therefore, specifications such as the following are *incorrect*:

```
TABLE HQ.T_*, TARGET RPT.T_*;  
MAP HQ.T_*, TARGET RPT.T_*;
```

The preceding mappings produce incorrect results, because the wildcard in the target specification is replaced with `T_TEST` (the name of a source object), making the whole target name `T_T_TESTn`. The following illustrates the incorrect results:

- `HQ.T_TEST1` maps to `RPT.T_T_TEST1`
- `HQ.T_TEST2` maps to `RPT.T_T_TEST2`
- (The same pattern applies to all other `HQ.T_TESTn` mappings.)

The following examples show the correct use of asterisk wildcards.

```
MAP HQ.T_*, TARGET RPT.*;
```

The preceding example produces the following correct results:

- `HQ.T_TEST1` maps to `RPT.T_TEST1`
- `HQ.T_TEST2` maps to `RPT.T_TEST2`
- (The same pattern applies to all other `HQ.T_TESTn` mappings.)

Fallback Name Mapping

Oracle GoldenGate has a fallback mapping mechanism in the event that a source name cannot be mapped to a target name. If an exact match cannot be found on the target for a case-sensitive source object, Replicat tries to map the source name to the same name in upper or lower case (depending on the database type) on the target. Fallback name mapping is controlled by the `NAMEMATCH` parameters. For more information, see *Reference for Oracle GoldenGate*.

Wildcard Mapping from Pre-11.2.1 Trail Version

If Replicat is configured to read from a trail file that is a version prior to Oracle GoldenGate 11.2.1, the target mapping is made in the following manner to provide backward compatibility.

- Quoted object names are case-sensitive.
- Unquoted object names are case-insensitive.

The following maps a case-sensitive table name "abc" to target "abc". This only happens with a trail that was written by pre-11.2.1 Extract for SQL Server databases with a case-sensitive configuration. In this example, if the target database is Oracle

Database or DB2 fallback name mapping is performed if the target database does not contain case-sensitive "abc" but does have table ABC. (See [Fallback Name Mapping](#).)

```
MAP hq."abc", TARGET hq.*;
```

The following example maps a case-insensitive table name abc to target table name ABC. Previous releases of Oracle GoldenGate stored case-insensitive object names to the trail in upper case; thus the target table name is always upper cased. For case-insensitive name conversion, the comparison is in uppercase, A to Z characters only, in US-ASCII without taking locale into consideration.

```
MAP hq.abc, TARGET hq.*;
```

Asterisks or Question Marks as Literals in Object Names

If the name of an object itself includes an asterisk or a question mark, the entire name must be escaped and placed within double quotes, as in the following example:

```
TABLE HT."\"?ABC";
```

How Wildcards are Resolved

By default, when an object name is wildcarded, the resolution for that object occurs when the first row from the source object is processed. (By contrast, when the name of an object is stated explicitly, its resolution occurs at process startup.) To change the rules for resolving wildcards, use the `WILDCARDRESOLVE` parameter. The default is `DYNAMIC`.

Excluding Objects from a Wildcard Specification

You can combine the use of wildcard object selection with explicit object exclusion by using the `EXCLUDEWILDCARDOBJECTSONLY`, `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `MAPEXCLUDE`, and `TABLEEXCLUDE` parameters. See *Reference for Oracle GoldenGate* for descriptions and syntax.

Differentiating Case-Sensitive Column Names from Literals

By default, Oracle GoldenGate follows SQL-92 rules for specifying column names and literals. In Oracle GoldenGate parameter files, conversion functions, user exits, and commands, case-sensitive column names must be enclosed within double quotes if the database requires quotes around a name to support case-sensitivity. For example:

```
"columnA"
```

Case-sensitive column names in databases that do not require quotes to enforce case-sensitivity must be specified as they are stored in the database. For example:

```
ColumnA
```

Literals must be enclosed within single quotes. In the following example, `Product_Code` is a case-sensitive column name in an Oracle database, and the other strings are literals.

```
@CASE ("Product_Code", 'CAR', 'A car', 'TRUCK', 'A truck')
```

5

Using Oracle GoldenGate for Live Reporting

This chapter describes the usage of Oracle GoldenGate for live reporting.

Topics:

- [Overview of the Reporting Configuration](#)
- [Creating a Standard Reporting Configuration](#)
- [Creating a Reporting Configuration with a Data Pump on the Source System](#)
- [Creating a Reporting Configuration with a Data Pump on an Intermediary System](#)
- [Creating a Cascading Reporting Configuration](#)

Overview of the Reporting Configuration

The most basic Oracle GoldenGate configuration is a one-to-one configuration that replicates in one direction: from a source database to a target database that is used only for data retrieval purposes such as reporting and analysis. Oracle GoldenGate supports like-to-like or heterogeneous transfer of data, with capabilities for filtering and conversion on either system in the configuration (support varies by database platform).



Oracle GoldenGate supports different reporting topologies that enable you to custom-configure the processes based on your requirements for scalability, availability, and performance. This section contains things to take into consideration when choosing a reporting configuration.

- [Filtering and Conversion](#)
- [Read-only vs. High Availability](#)
- [Additional Information](#)

Filtering and Conversion

Data filtering and data conversion both add overhead, and these activities are sometimes prone to configuration errors. If Oracle GoldenGate must perform a large amount of filtering and conversion, consider using one or more data pumps to handle this work. You can use Replicat for this purpose, but you would be sending more data across the network that way, as it will be unfiltered. You can split filtering and

conversion between the two systems by dividing it between the data pump and Replicat.

To filter data, you can use:

- A `FILTER` or `WHERE` clause in a `TABLE` statement (Extract) or in a `MAP` statement (Replicat)
- A SQL query or procedure
- User exits

To transform data, you can use:

- The Oracle GoldenGate conversion functions
- A user exit from the Extract or Replicat process that applies rules from an external transformation solution, then returns the manipulated data to Oracle GoldenGate
- Replicat to deliver data directly to an ETL solution or other transformation engine

For more information about Oracle GoldenGate's filtering and conversion support, see:

- [Mapping and Manipulating Data](#)
- [Customizing Oracle GoldenGate Processing](#)

Read-only vs. High Availability

The Oracle GoldenGate live reporting configuration supports a read-only target. See [Configuring Oracle GoldenGate for Active-Active High Availability](#) if the target in this configuration will also be used for transactional activity in support of high availability.

Additional Information

The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

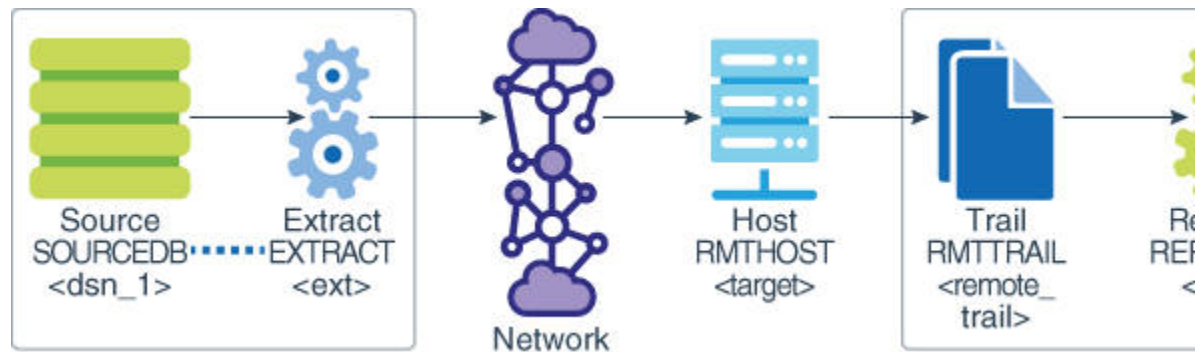
- For additional system requirements, process configuration, and database setup requirements, see the Oracle GoldenGate installation and configuration document for your database type. These guides are listed in the [Preface](#) of this book.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see [Configuring Online Change Synchronization](#).
- For additional tuning options for Oracle GoldenGate, see [Tuning the Performance of Oracle GoldenGate](#).
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Standard Reporting Configuration

In the standard Oracle GoldenGate configuration, one Extract group sends captured data over TCP/IP to a trail on the target system, where it is stored until processed by one Replicat group.

Refer to [Figure 5-1](#) for a visual representation of the objects you will be creating.

Figure 5-1 Configuration Elements for Creating a Standard Reporting Configuration



- Source System
- Target System

Source System

Configure the Manager process and Extract group on the source system.

To Configure the Manager Process

On the source, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).

To Configure the Extract Group

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext`.

```
ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail to be created on the target system.

```
ADD RMTTRAIL remote_trail, EXTRACT ext
```

Use the `EXTRACT` argument to link this trail to the Extract group.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all of the supplementally logged columns if using integrated Replicat
LOGALLSUPCOLS
-- Valid for Oracle. Specify the name or IP address of the target system and
-- optional encryption across TCP/IP:
RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the target system:
```

```

ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;

```

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process

1. On the target, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Replicat Group

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions. All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep*.

```

ADD REPLICAT rep
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail
, BEGIN time

```

Use the `EXTTRAIL` argument to link the Replicat group to the remote trail.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```

-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;

```

 **Note:**

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPf` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES (*AFTER)` option can be used with `STRJRNPf`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Creating a Reporting Configuration with a Data Pump on the Source System

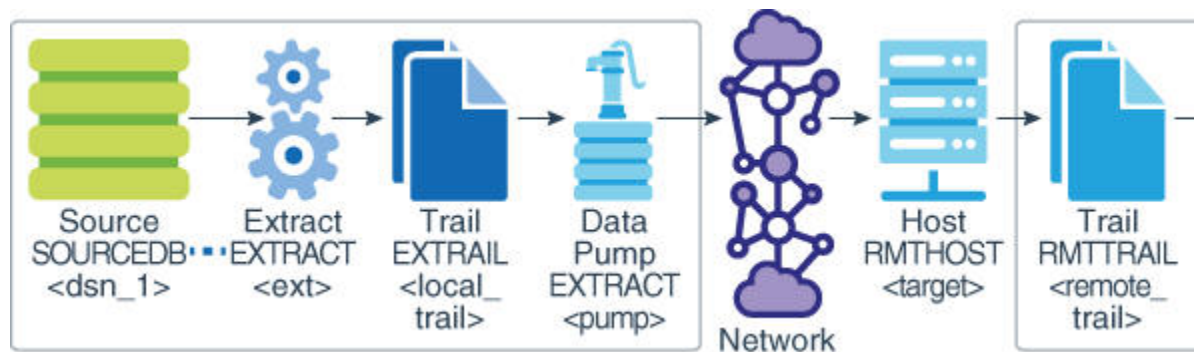
You can add a data pump on the source system to isolate the primary Extract from TCP/IP functions, to add storage flexibility, and to offload the overhead of filtering and conversion processing from the primary Extract.

In this configuration, the primary Extract writes to a local trail on the source system. A local data pump reads that trail and moves the data to a remote trail on the target system, which is read by Replicat.

You can, but are not required to, use a data pump to improve the performance and fault tolerance of Oracle GoldenGate.

Refer to [Figure 5-2](#) for a visual representation of the objects you will be creating.

Figure 5-2 Configuration Elements for Replicating to One Target with a Data Pump



- [Source System](#)
- [Target System](#)

Source System

Configure the Manager process and Extract group on the source system.

To Configure the Manager Process

1. On the source, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Primary Extract Group

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext*.

```
ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail. The primary Extract writes to this trail, and the data-pump Extract reads it.

```
ADD EXTTRAIL local_trail, EXTRACT ext
```

Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump group reads it.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][,USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to and
-- encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pump Extract Group

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump*.

```
ADD EXTRACT pump, EXTTRAILSOURCE local_trail, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the target system.

```
ADD RMTTRAIL remote_trail, EXTRACT pump
```

Use the `EXTRACT` argument to link the remote trail to the data pump group. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the target system:
ENCRYPTTRAIL alogrithm
RMTTRAIL remote_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process

1. On the target, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Replicat Group

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep*.

```
ADD REPLICAT rep
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail
, BEGIN time
```

Use the `EXTTRAIL` argument to link the Replicat group to the remote trail.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

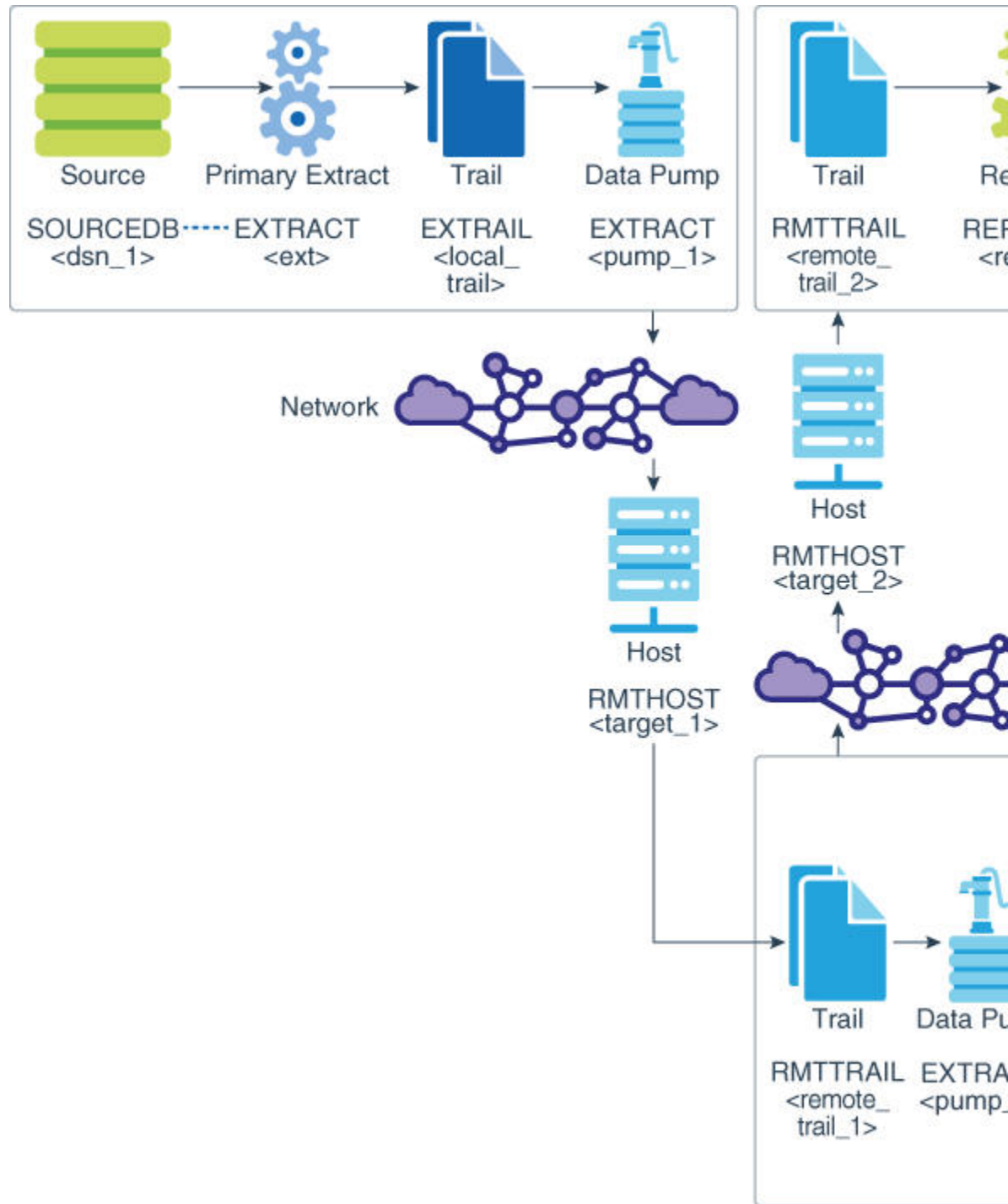
Note:

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Creating a Reporting Configuration with a Data Pump on an Intermediary System

You can use an intermediary system as a transfer point between the source and target systems. In this configuration, a data pump on the source system sends captured data to a remote trail on the intermediary system. A data pump on the intermediary system reads the trail and sends the data to a remote trail on the target. A Replicat on the target reads the remote trail and applies the data to the target database.

Figure 5-3 Configuration Elements for Replication through an Intermediary System



When considering this topology, take note of the following:

- This configuration is practical if the source and target systems are in different networks and there is no direct connection between them. You can transfer the data through an intermediary system that can connect to both systems.

- This configuration can be used to add storage flexibility to compensate for deficiencies on the source or target.
- This configuration can be used to perform data filtering and conversion if the character sets on all systems are identical. If character sets differ, the data pump cannot perform conversion between character sets, and you must configure Replicat to perform the conversion and transformation on the target.
- To use the data pump on the intermediary system to perform data conversion and transformation, assuming character sets are identical, you must create a source definitions file and a target definitions file with the DEFGEN utility and then transfer both files to the intermediary system. See [Associating Replicated Data with Metadata](#) for more information about definitions files and conversion.
- This configuration is a form of cascaded replication. However, in this configuration, data is not applied to a database on the intermediary system. See [Creating a Cascading Reporting Configuration](#) to include a database on the intermediary system in the Oracle GoldenGate configuration.
- [Source System](#)
- [Intermediary System](#)
- [Target System](#)

Source System

Refer to Figure 10 for a visual representation of the objects you will be creating.

To Configure the Manager Process

1. On the source, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Primary Extract Group on the Source

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext`.

```
ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail. The primary Extract writes to this trail, and the data-pump Extract reads it.

```
ADD EXTTRAIL local_trail, EXTRACT ext
```

Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump group reads it.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to and
-- encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pump on the Source

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_1*.

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail. For a local Extract, you must use `EXTTRAIL` not `RMTTRAIL`.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the intermediary system.

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
```

Use the `EXTRACT` argument to link the remote trail to the *pump_1* data pump group. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the *pump_1* data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the intermediary system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on intermediary system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Intermediary System

Configure the Manager process and data pump on the intermediary system.

To Configure the Manager Process on the Intermediary System

1. On the intermediary system, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Data Pump on the Intermediary System

1. On the intermediary system, use the `ADD EXTRACT` command to create a data-pump group. For documentation purposes, this group is called `pump_2`.

```
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_1, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the trail that you created on this system

2. On the intermediary system, use the `ADD RMTTRAIL` command to specify a remote trail on the target system.

```
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

Use the `EXTRACT` argument to link the remote trail to the `pump_2` data pump. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the intermediary system, use the `EDIT PARAMS` command to create a parameter file for the `pump_2` data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_2
-- Note that no database login parameters are required in this case.
-- Specify the target definitions file if SOURCEDEFS was used:
TARGETDEFS full_pathname
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the target system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

•

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process on the Target

1. On the target system, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).

2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group on the Target

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called `rep`.

```
ADD REPLICAT rep
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2,
, BEGIN time
```

Use the `EXTTRAIL` argument to link the Replicat group to the trail on this system.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.|owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Note:

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPf` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPf`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Creating a Cascading Reporting Configuration

Oracle GoldenGate supports cascading synchronization, where Oracle GoldenGate propagates data changes from the source database to a second database, and then on to a third database. In this configuration:

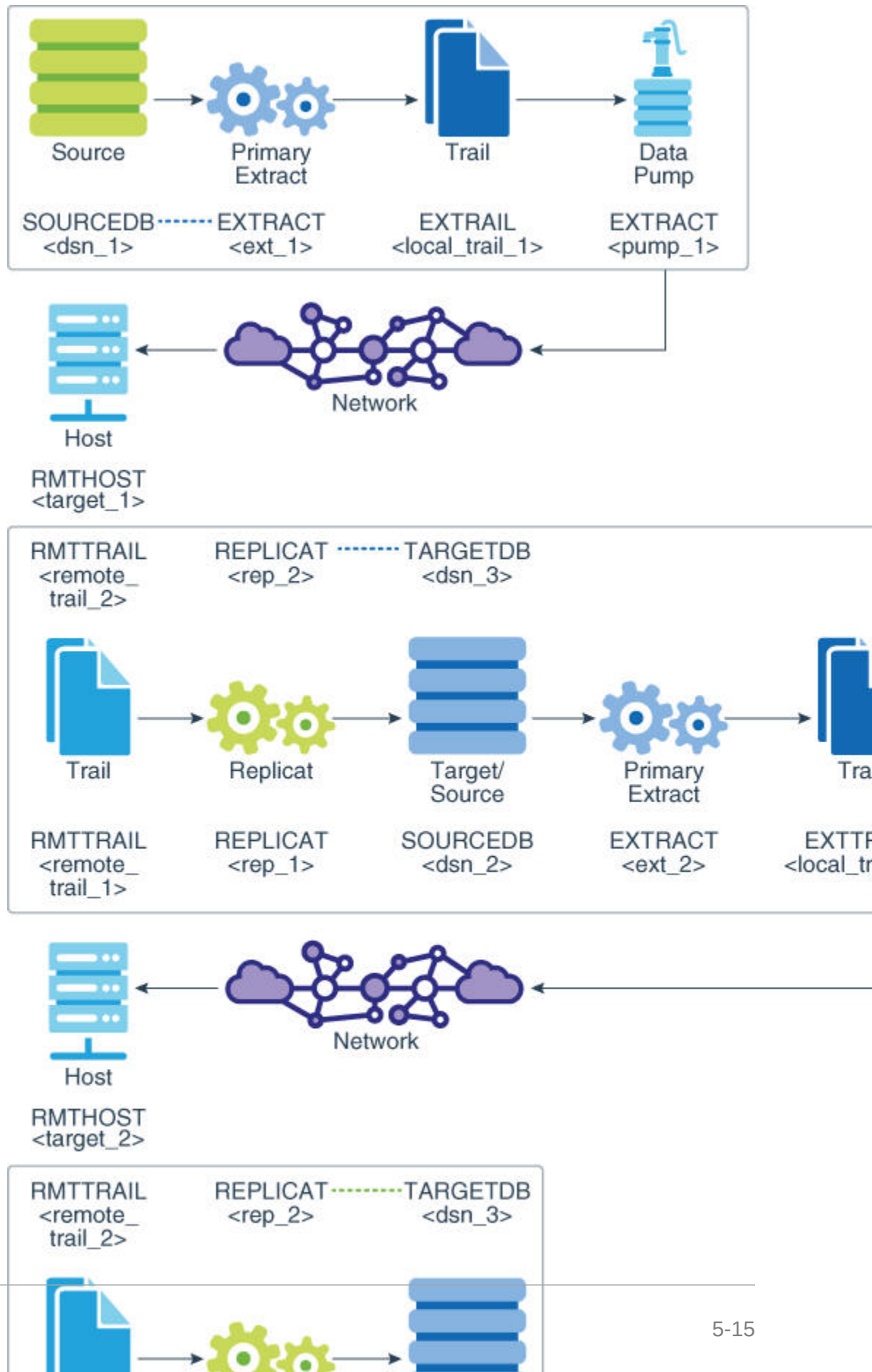
- A primary Extract on the source writes captured data to a local trail, and a data pump sends the data to a remote trail on the second system in the cascade.

- On the second system, Replicat applies the data to the local database.
- Another primary Extract on that same system captures the data from the local database and writes it to a local trail.
- A data pump sends the data to a remote trail on the third system in the cascade, where it is applied to the local database by another Replicat.

 **Note:**

See [Creating a Reporting Configuration with a Data Pump on an Intermediary System](#) if you do not need to apply the replicated changes to a database on the secondary system.

Figure 5-4 Cascading Configuration



Use this configuration if:

- One or more of the target systems does not have a direct connection to the source, but the second system can connect in both directions.
- You want to limit network activity from the source system.
- You are sending data to two or more servers that are very far apart geographically, such as from Chicago to Los Angeles and then from Los Angeles to servers throughout China.

When considering this topology, take note of the following:

- This configuration can be used to perform data filtering and conversion if the character sets on all systems are identical. If character sets differ, a data pump cannot perform conversion between character sets, and you must configure Replicat to perform the conversion and transformation on the target.
- To use the data pump on the second system to perform data conversion and transformation, assuming character sets are identical, you must create a source definitions file on the first system with the DEFGEN utility and then transfer it to the second system. Additionally, you must create a source definitions file on the second system and transfer it to the third system. See [Associating Replicated Data with Metadata](#) for more information about definitions files and conversion.
- On the second system, you must configure the Extract group to capture Replicat activity and to ignore local business application activity. The Extract parameters that control this behavior are `IGNOREAPPLOPS` and `GETREPLICATES`.
- [Source System](#)
- [Second System in the Cascade](#)
- [Third System in the Cascade](#)

Source System

Refer to [Figure 5-4](#) for a visual representation of the objects you will be creating.

To Configure the Manager Process on the Source

1. On the source, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Primary Extract Group on the Source

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext_1`.

```
ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail.

```
ADD EXTTRAIL local_trail_1, EXTRACT ext_1
```

Use the `EXTRACT` argument to link this trail to the `ext_1` Extract group.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the `ext_1` Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pump on the Source

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called `pump_1`.

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the second system in the cascade.

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
```

Use the `EXTRACT` argument to link the remote trail to the `pump_1` data pump group. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the `pump_1` data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information if using NOPASSTHROUGH:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of second system in cascade
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the second system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```


Second System in the Cascade

Configure the Manager process, Replicat group, and data pump on the second system in the cascade.

To Configure the Manager Process on the Second System

1. On the second system, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group on the Second System

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On the second system, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called `rep_1`.

```
ADD REPLICAT rep_1
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1,
, BEGIN time
```

Use the `EXTTRAIL` option to link the `rep_1` group to the remote trail `remote_trail_1` that is on the local system.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep_1
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

 **Note:**

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPf` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPf`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

To Configure an Extract Group on the Second System

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext_2`.

```
ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the second system, use the `ADD EXTTRAIL` command to specify a local trail that will be created on the third system.

```
ADD EXTTRAIL local_trail_2, EXTRACT ext_2
```

Use the `EXTRACT` argument to link this local trail to the `ext_2` Extract group.

3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the `ext_2` Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_2
-- Ignore local DML, capture Replicat DML:
IGNOREAPPLOPS, GETREPLICATES
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

 **Note:**

If replicating DDL operations, `IGNOREAPPLOPS`, `GETREPLICATES` functionality is controlled by the `DDLOPTIONS` parameter.

To Configure the Data Pump on the Second System

1. On the second system, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_2*.

```
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.

2. On the second system, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the third system in the cascade.

```
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

Use the `EXTRACT` argument to link the remote trail to the *pump_2* data pump group. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the *pump_2* data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_2
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of third system in cascade
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the third system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Third System in the Cascade

Configure the Manager process and Replicat group on the third system in the cascade.

To Configure the Manager Process

1. On the third system, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group

1. On the third system, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On the third system, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep_2*.

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2,
, BEGIN time
```

Use the `EXTTRAIL` option to link the `rep_2` group to the `remote_trail_2` trail.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the third system, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep_2
-- Specify database login information as needed for the database:
[TARGETDB dsn_3][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Note:

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPf` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPf`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

6

Using Oracle GoldenGate for Real-time Data Distribution

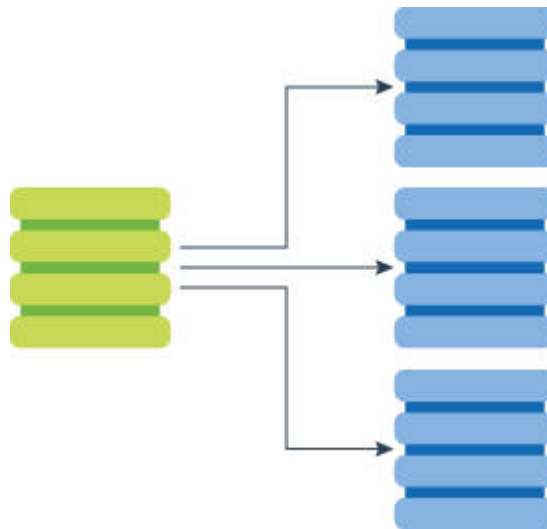
This chapter describes the usage of Oracle GoldenGate for real-time data distribution.

Topics:

- [Overview of the Data-distribution Configuration](#)
- [Considerations for a Data-distribution Configuration](#)
- [Creating a Data Distribution Configuration](#)

Overview of the Data-distribution Configuration

A data distribution configuration is a one-to-many configuration. Oracle GoldenGate supports synchronization of a source database to any number of target systems. Oracle GoldenGate supports like-to-like or heterogeneous transfer of data, with capabilities for filtering and conversion on any system in the configuration (support varies by database platform).



Considerations for a Data-distribution Configuration

These sections describe considerations for a data-distribution configuration.

- [Fault Tolerance](#)
- [Filtering and Conversion](#)
- [Read-only vs. High Availability](#)

- [Additional Information](#)

Fault Tolerance

For a data distribution configuration, the use of data pumps on the source system ensures that if network connectivity to any of the targets fails, the captured data still can be sent to the other targets. Use a primary Extract group and one data-pump Extract group for each target.

Filtering and Conversion

You can use any process to perform filtering and conversion. However, using the data pumps to perform filtering operations removes that processing overhead from the primary Extract group, and it reduces the amount of data that is sent across the network. See [Mapping and Manipulating Data](#) for filtering and conversion options.

Read-only vs. High Availability

The data distribution configuration supports read-only targets. See [Configuring Oracle GoldenGate for Active-Active High Availability](#) if any target in this configuration will also be used for transactional activity in support of high availability.

Additional Information

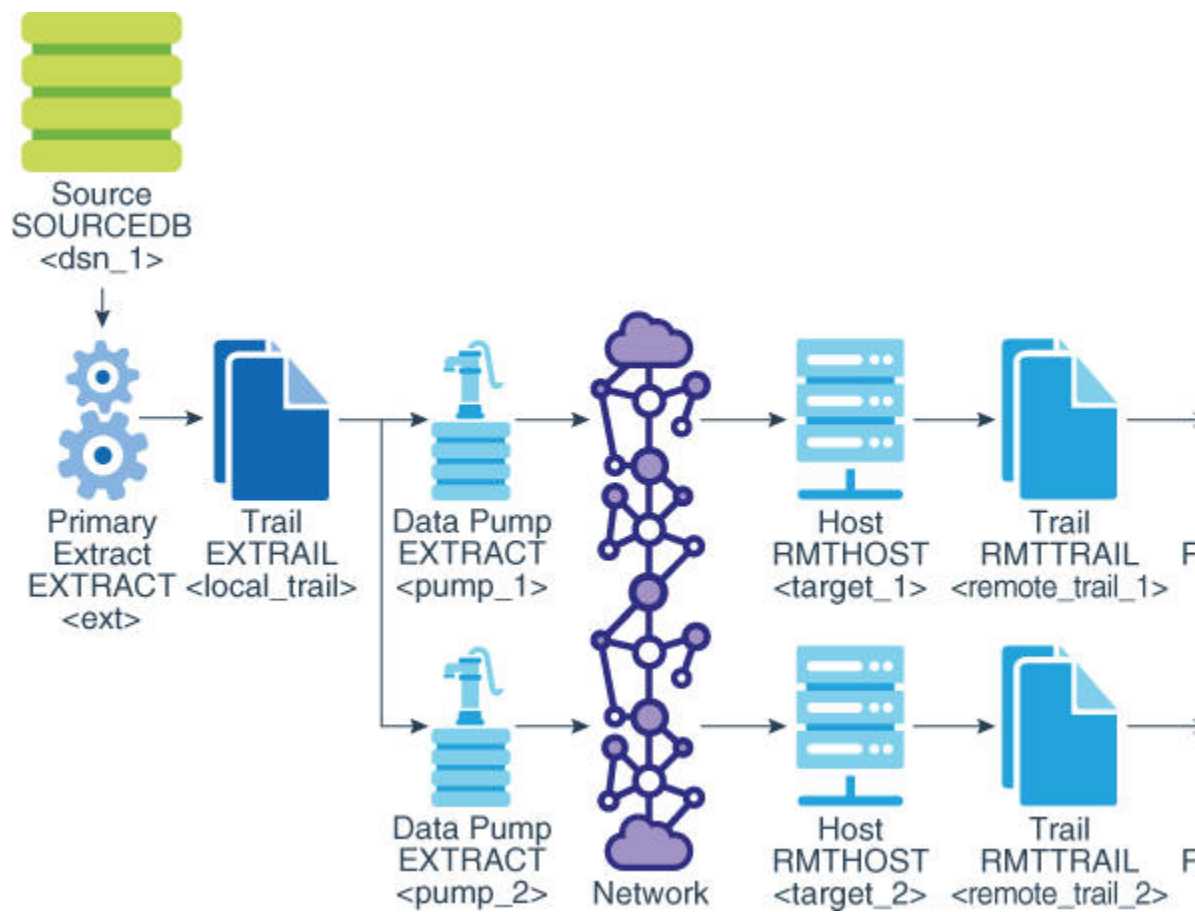
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see the Oracle GoldenGate installation and configuration document for your database type. These guides are listed in the [Preface](#) of this book.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see [Configuring Online Change Synchronization](#).
- For additional tuning options for Oracle GoldenGate, see [Tuning the Performance of Oracle GoldenGate](#).
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Data Distribution Configuration

Refer to [Figure 6-1](#) for a visual representation of the objects you will be creating.

Figure 6-1 Oracle GoldenGate Configuration Elements for Data Distribution



- Source System
- Target Systems

Source System

Configure the Manager process and primary Extract on the source system.

To Configure the Manager Process

1. On the source, configure the Manager process. See [Configuring Manager and Network Communications](#) for instructions.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Primary Extract

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext`.

```
ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail.

```
ADD EXTTRAIL local_trail, EXTRACT ext
```

Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump groups read it

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Use `EXTTRAIL` to specify the local trail.

To Configure the Data Pump Extract Groups

1. On the source, use the `ADD EXTRACT` command to create a data pump for each target system. For documentation purposes, these groups are called *pump_1* and *pump_2*.

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail, BEGIN time
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and supply the name of the local trail.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on each of the target systems.

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

Use the `EXTRACT` argument to link each remote trail to a different data pump group. The linked data pump writes to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for each of the data pumps. Include the following parameters plus any others that apply to your database environment.

Parameter file for *pump_1*:

```
-- Identify the data pump group:
EXTRACT pump_1
```



```
-- Specify database login information:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the first target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on first target system:
ENCRYPTTRAIL algorithm
RMTRAIL remote_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Parameter file for *pump_2*:

```
-- Identify the data pump group:
EXTRACT pump_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the second target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on second target system:
ENCRYPTTRAIL algorithm
RMTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Target Systems

Configure the Manager process and Replicat groups on the target systems.

To Configure the Manager Process

1. On each target, configure the Manager process. See [Configuring Manager and Network Communications](#) for instructions.
2. In each Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Groups

1. On each target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On each target, use the `ADD REPLICAT` command to create a Replicat group for the remote trail on that system. For documentation purposes, these groups are called *rep_1* and *rep_2*.

Command on *target_1*:

```
ADD REPLICAT rep_1
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1, BEGIN time
```

Command on *target_2*:

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2, BEGIN time
```

Use the `EXTTRAIL` argument to link the Replicat group to the correct trail.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On each target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Use the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

Parameter file for *rep_1*:

```
-- Identify the Replicat group:
REPLICAT rep_1
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.|owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Parameter file for *rep_2*:

```
-- Identify the Replicat group:
REPLICAT rep_2
-- Specify database login information as needed for the database:
[TARGETDB dsn_3][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.|owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

You can use any number of `MAP` statements for any given Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

7

Configuring Oracle GoldenGate for Real-time Data Warehousing

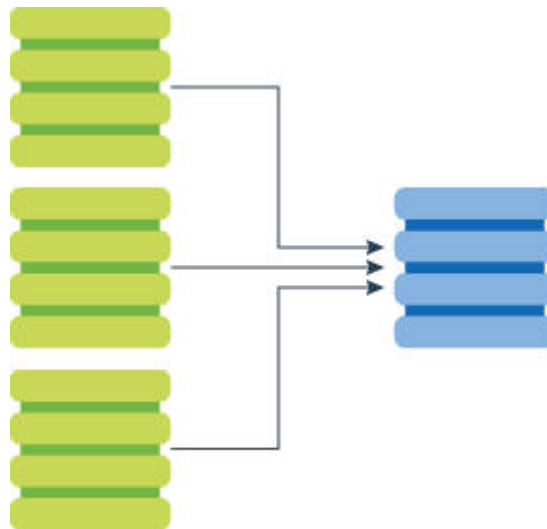
This chapter describes how to configure Oracle GoldenGate for real-time data warehousing.

Topics:

- [Overview of the Data Warehousing Configuration](#)
- [Considerations for a Data Warehousing Configuration](#)
- [Creating a Data Warehousing Configuration](#)

Overview of the Data Warehousing Configuration

A data warehousing configuration is a many-to-one configuration. Multiple source databases send data to one target warehouse database. Oracle GoldenGate supports like-to-like or heterogeneous transfer of data, with capabilities for filtering and conversion on any system in the configuration (support varies by database platform).



Considerations for a Data Warehousing Configuration

This section describes considerations for a data warehousing configuration.

- [Isolation of Data Records](#)
- [Data Storage](#)
- [Filtering and Conversion](#)

- [Additional Information](#)

Isolation of Data Records

This configuration assumes that each source database contributes different records to the target system. If the same record exists in the same table on two or more source systems and can be changed on any of those systems, conflict resolution routines are needed to resolve conflicts when changes to that record are made on both sources at the same time and replicated to the target table. See [Configuring Oracle GoldenGate for Active-Active High Availability](#) for more information about resolving conflicts.

Data Storage

You can divide the data storage between the source systems and the target system to reduce the need for massive amounts of disk space on the target system. This is accomplished by using a data pump on each source, rather than sending data directly from each Extract across the network to the target.

- A primary Extract writes to a local trail on each source.
- A data-pump Extract on each source reads the local trail and sends it across TCP/IP to a dedicated Replicat group.

Filtering and Conversion

If not all of the data from a source system will be sent to the data warehouse, you can use the data pump to perform the filtering. This removes that processing overhead from the primary Extract group, and it reduces the amount of data that is sent across the network. See [Mapping and Manipulating Data](#) for filtering and conversion options.

Additional Information

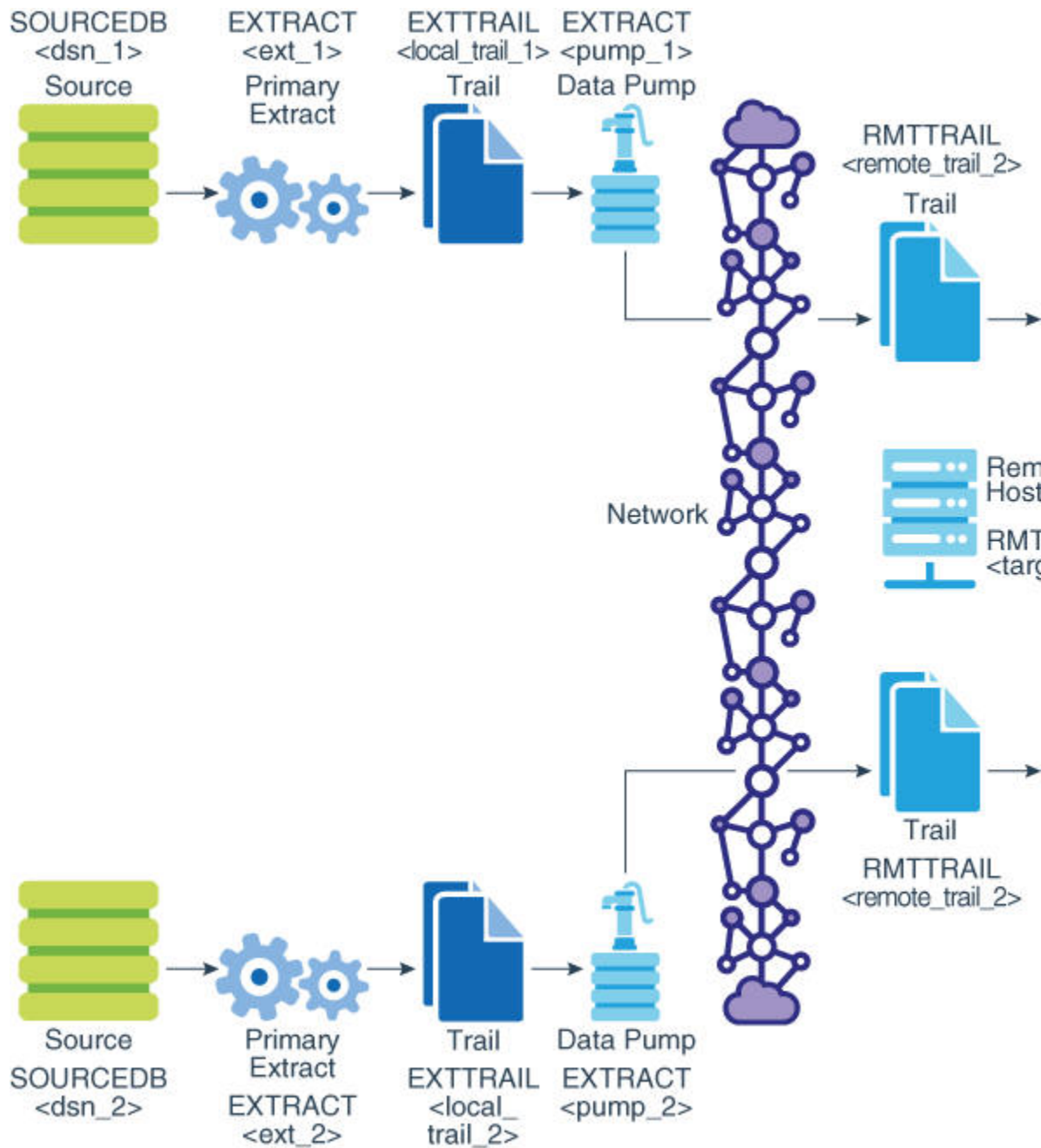
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see the Oracle GoldenGate installation and configuration document for your database type. These guides are listed in the [Preface](#) of this book.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see [Configuring Online Change Synchronization](#).
- For additional tuning options for Oracle GoldenGate, see [Tuning the Performance of Oracle GoldenGate](#).
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Data Warehousing Configuration

Refer to [Figure 7-1](#) for a visual representation of the objects you will be creating.

Figure 7-1 Configuration for Data Warehousing



- Source Systems
- Target System

Source Systems

Configure the Manager process and primary Extract groups for the source systems.

To Configure the Manager Process

1. On each source, configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).
2. In each Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail on the local system.

To Configure the primary Extract Groups

1. On each source, use the `ADD EXTRACT` command to create a primary Extract group. For documentation purposes, these groups are called `ext_1` and `ext_2`.

Command on *source_1*:

```
ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

Command on *source_2*:

```
ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On each source, use the `ADD EXTTRAIL` command to create a local trail.

Command on *source_1*:

```
ADD EXTTRAIL local_trail_1, EXTRACT ext_1
```

Command on *source_2*:

```
ADD EXTTRAIL local_trail_2, EXTRACT ext_2
```

Use the `EXTRACT` argument to link each Extract group to the local trail on the same system. The primary Extract writes to this trail, and the data-pump reads it.

3. On each source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

Parameter file for *ext_1*:

```
-- Identify the Extract group:
EXTRACT ext_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Parameter file for *ext_2*:

```
-- Identify the Extract group:
EXTRACT ext_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat or CDR
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pumps

1. On each source, use the `ADD EXTRACT` command to create a data pump Extract group. For documentation purposes, these pumps are called *pump_1* and *pump_2*.

Command on *source_1*:

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
```

Command on *source_2*:

```
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the trail on the local system

2. On each source, use the `ADD RMTTRAIL` command to create a remote trail on the target.

Command on *source_1*:

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
```

Command on *source_2*:

```
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

Use the `EXTRACT` argument to link each remote trail to a different data pump. The data pump writes to this trail over TCP/IP, and a Replicat reads from it.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On each source, use the `EDIT PARAMS` command to create a parameter file for the data pump group. Include the following parameters plus any others that apply to your database environment.

Parameter file for *pump_1*:

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
```

```
-- Specify the remote trail and encryption algorithm on the target system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Parameter file for *pump_2*:

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the target system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Target System

Configure the Manager process and primary Replicat groups for the target system.

To Configure the Manager Process

1. Configure the Manager process. See [Configuring Manager and Network Communications](#) for instructions.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Groups

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each remote trail that you created. For documentation purposes, these groups are called *rep_1* and *rep_2*.

Command to add *rep_1*:

```
ADD REPLICAT rep_1
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1, BEGIN time
```

Command to add *rep_2*:

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2, BEGIN time
```

Use the `EXTTRAIL` argument to link the Replicat group to the trail.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for each Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

Parameter file for `rep_1`:

```
-- Identify the Replicat group:
REPLICAT rep_1
-- Specify database login information as needed for the database:
[TARGETDB dsn_3][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.|owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Parameter file for `rep_1`:

```
-- Identify the Replicat group:
REPLICAT rep_2
-- Specify database login information as needed for the database:
[TARGETDB dsn_3][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.|owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

You can use any number of `MAP` statements for any given Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

8

Configuring Oracle GoldenGate to Maintain a Live Standby Database

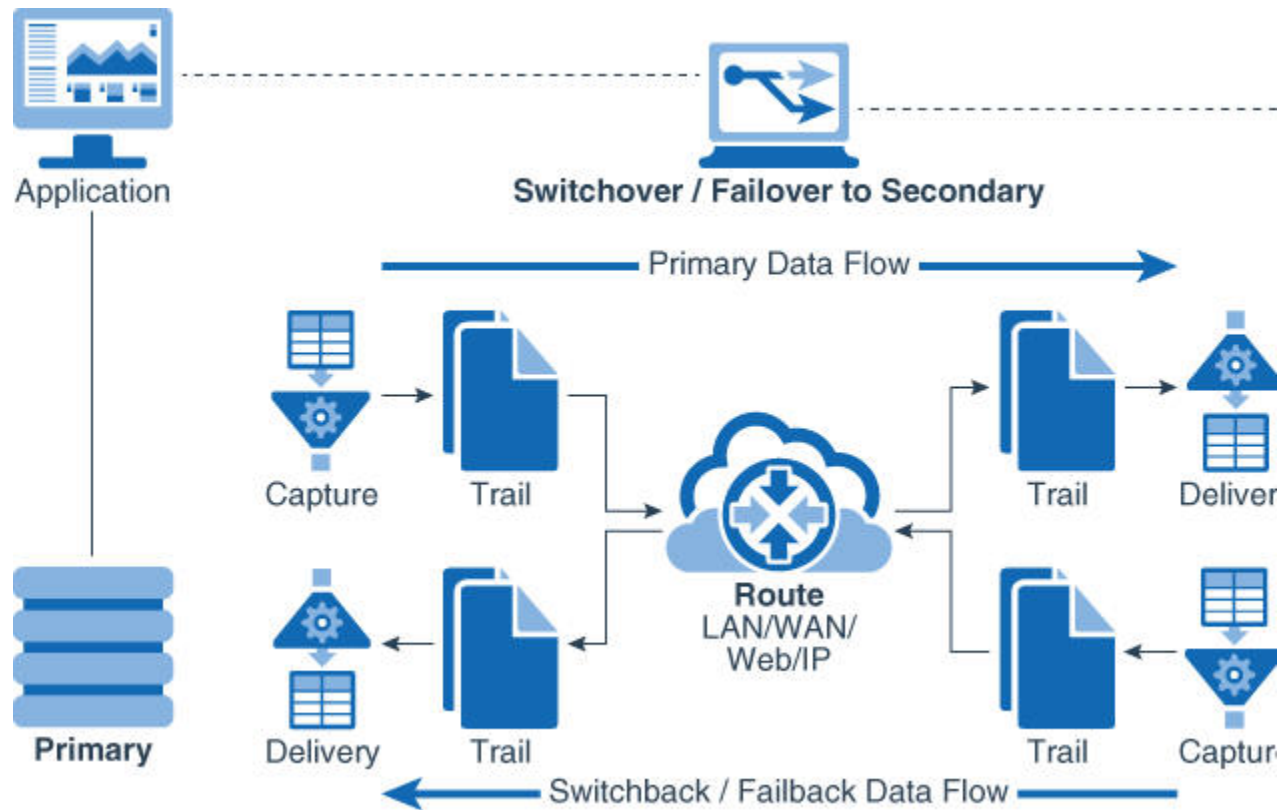
This chapter describes how to configure Oracle GoldenGate to maintain a live standby database.

Topics:

- [Overview of a Live Standby Configuration](#)
- [Considerations for a Live Standby Configuration](#)
- [Creating a Live Standby Configuration](#)
- [Configuration from Standby to Active Source](#)
- [Moving User Activity in a Planned Switchover](#)
- [Moving User Activity in an Unplanned Failover](#)

Overview of a Live Standby Configuration

Oracle GoldenGate supports an active-passive bi-directional configuration, where Oracle GoldenGate replicates data from an active primary database to a full replica database on a live standby system that is ready for failover during planned and unplanned outages.



In this configuration, there is an inactive Oracle GoldenGate Extract group and an inactive data pump on the live standby system. Both of those groups remain stopped until just before user applications are switched to the live standby system in a switchover or failover. When user activity moves to the standby, those groups begin capturing transactions to a local trail, where the data is stored on disk until the primary database can be used again.

In the case of a failure of the primary system, the Oracle GoldenGate Manager and Replicat processes work in conjunction with a database instantiation taken from the standby to restore parity between the two systems after the primary system is recovered. At the appropriate time, users are moved back to the primary system, and Oracle GoldenGate is configured in ready mode again, in preparation for future failovers.

Considerations for a Live Standby Configuration

These sections describe considerations for a live standby configuration.

- [Trusted Source](#)
- [Duplicate Standby](#)
- [DML on the Standby System](#)
- [Oracle GoldenGate Processes](#)
- [Backup Files](#)
- [Failover Preparedness](#)

- [Sequential Values that are Generated by the Database](#)
- [Additional Information](#)

Trusted Source

The primary database is the *trusted source*. This is the database that is the *active source* during normal operating mode, and it is the one from which the other database is derived in the initial synchronization phase and in any subsequent resynchronizations. Maintain frequent backups of the trusted source data.

Duplicate Standby

In most implementations of a live standby, the source and target databases are identical in content and structure. Data mapping, conversion, and filtering typically are not appropriate practices in this kind of configuration, but Oracle GoldenGate does support such functionality if required by your business model. To support these functions, use the options of the `TABLE` and `MAP` parameters.

DML on the Standby System

If your applications permit, you can use the live standby system for reporting and queries, but not DML. If there will be active transactional applications on the live standby system that affect objects in the Oracle GoldenGate configuration, you should configure this as an active-active configuration. See [Configuring Oracle GoldenGate for Active-Active High Availability](#) for more information.

Oracle GoldenGate Processes

During normal operating mode, leave the primary Extract and the data pump on the live standby system stopped, and leave the Replicat on the active source stopped. This prevents any DML that occurs accidentally on the standby system from being propagated to the active source. Only the Extract, data pump, and Replicat that move data from the active source to the standby system can be active.

Backup Files

Make regular backups of the Oracle GoldenGate working directories on the primary and standby systems. This backup must include all of the files that are installed at the root level of the Oracle GoldenGate installation directory and all of the sub-directories within that directory. Having a backup of the Oracle GoldenGate environment means that you will not have to recreate your process groups and parameter files.

Failover Preparedness

Make certain that the primary and live standby systems are ready for immediate user access in the event of a planned switchover or an unplanned source failure. The following components of a high-availability plan should be made easily available for use on each system:

- Scripts that grant insert, update, and delete privileges.
- Scripts that enable triggers and cascaded delete constraints on the live standby system. (These may have been disabled during the setup procedures that were

outlined in the Oracle GoldenGate installation and configuration document for your database type.)

- Scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.
- A failover procedure for moving users to the live standby if the source system fails.

Sequential Values that are Generated by the Database

If database-generated values, such as Oracle sequences, are used as part of a key, the range of values must be different on each system, with no chance of overlap. If the application permits, you can add a location identifier to the value to enforce uniqueness.

For Oracle databases, Oracle GoldenGate can be configured to replicate sequences in a manner that ensures uniqueness on each database. To replicate sequences, use the `SEQUENCE` and `MAP` parameters. For more information, see *Reference for Oracle GoldenGate*.

Additional Information

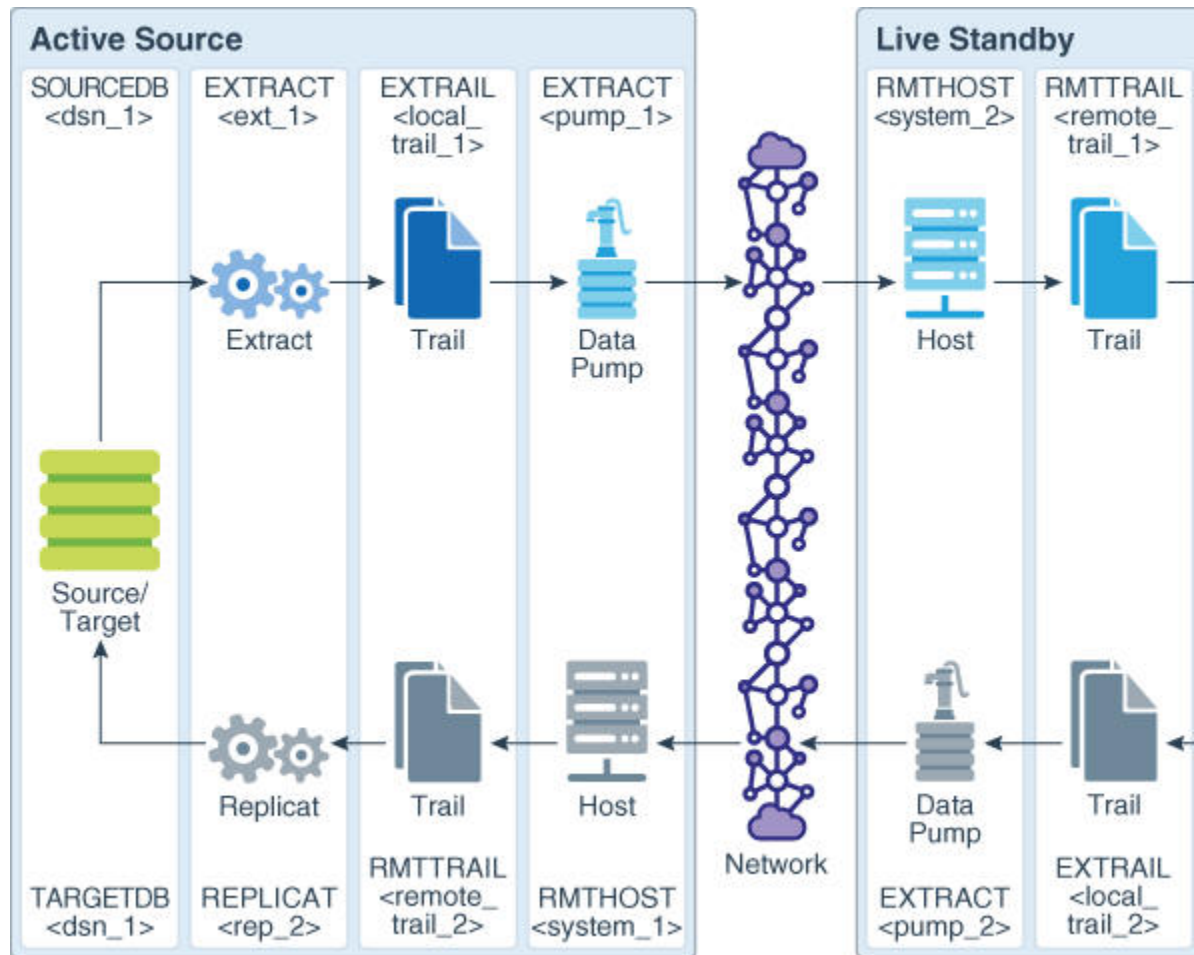
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see the Oracle GoldenGate installation and configuration document for your database type. These guides are listed in the [Preface](#) of this book.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see [Configuring Online Change Synchronization](#).
- For additional tuning options for Oracle GoldenGate, see [Tuning the Performance of Oracle GoldenGate](#).
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see *Reference for Oracle GoldenGate for Windows and UNIX*.

Creating a Live Standby Configuration

Refer to [Figure 8-1](#) for a visual representation of the objects you will be creating.

Figure 8-1 Oracle GoldenGate configuration elements for live standby



- [Prerequisites on Both Systems](#)
- [Configuration from Active Source to Standby](#)

Prerequisites on Both Systems

Perform the following prerequisites on both systems.

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). For instructions, see [Creating a Checkpoint Table](#).
2. Configure the Manager process according to the instructions in [Configuring Manager and Network Communications](#).

Configuration from Active Source to Standby

These steps configure Oracle GoldenGate to capture data from the primary database and replicate it to the standby database.

To Configure the Primary Extract Group

Perform these steps on the active source.

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext_1`.

```
ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called `local_trail_1`.

```
ADD EXTTRAIL local_trail_1, EXTRACT ext_1
```

For `EXTRACT`, specify the `ext_1` group to write to this trail.

3. Use the `EDIT PARAMS` command to create a parameter file for the `ext_1` group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_1
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Data Pump

Perform these steps on the active source.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called `pump_1`.

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
```

For `EXTTRAILSOURCE`, specify `local_trail_1` as the data source.

2. Use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the standby system.

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
```

For `EXTRACT`, specify the `pump_1` data pump to write to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. Use the `EDIT PARAMS` command to create a parameter file for the `pump_1` group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information as needed for the database:
```

```

[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the standby system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS system_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the standby system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;

```

To Configure the Replicat Group

Perform these steps on the live standby system.

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. Use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep_1*.

```

ADD REPLICAT rep_1
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1, BEGIN time

```

For EXTTRAIL, specify *remote_trail_1* as the trail that this Replicat reads.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. Use the EDIT PARAMS command to create a parameter file for the *rep_1* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```

-- Identify the Replicat group:
REPLICAT rep_1
-- State that source and target definitions are identical:
ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.]catalog.owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;

```

Configuration from Standby to Active Source

These steps configure Oracle GoldenGate in passive mode. In this mode, the Oracle GoldenGate processes are ready, but not started, to capture data from the secondary database and replicate it to the primary database after a switchover of transaction activity to the secondary system.

**Note:**

This is a reverse image of the configuration that you just created.

To Configure the Primary Extract Group

Perform these steps on the live standby system.

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called `ext_2`.

```
ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
```

See *Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. Start the `TRANLOG` Extract `ext_2`. Also see [Preventing Data Looping](#).
3. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called `local_trail_2`.

```
ADD EXTTRAIL local_trail_2, EXTRACT ext_2
```

For `EXTRACT`, specify the `ext_2` group to write to this trail.

4. Use the `EDIT PARAMS` command to create a parameter file for the `ext_2` group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail this Extract writes to and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_2
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Data Pump

Perform these steps on the live standby system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called `pump_2`.

```
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
```

For `EXTTRAILSOURCE`, specify `local_trail_2` as the data source.

2. Use the `ADD RMTTRAIL` command to add a remote trail `remote_trail_2` that will be created on the active source system.

```
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

For EXTRACT, specify the *pump_2* data pump to write to this trail.

See *Reference for Oracle GoldenGate* for additional ADD RMTTRAIL options.

3. Use the EDIT PARAMS command to create a parameter file for the *pump_2* group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the active source system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS system_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on active source system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Replicat Group

Perform these steps on the active source.

1. Use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep_2*.

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1, BEGIN time
```

For EXTTRAIL, specify *remote_trail_2* as the trail that this Replicat reads.

See *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

2. Use the EDIT PARAMS command to create a parameter file for the *rep_2* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep_2
-- State that source and target definitions are identical:
ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB dsn_1][, USERIDALIAS alias]
-- Handle collisions between failback data copy and replication:
HANDLECOLLISIONS
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Moving User Activity in a Planned Switchover

This procedure moves user application activity from a primary database to a live standby system in a planned, graceful manner so that system maintenance and other procedures that do not affect the databases can be performed on the primary system.

- [Moving User Activity to the Live Standby](#)
- [Moving User Activity Back to the Primary System](#)

Moving User Activity to the Live Standby

To move user activity to the live standby:

1. (Optional) If you need to perform system maintenance on the secondary system, you can do so now or at the specified time later in these procedures, after moving users from the secondary system back to the primary system. In either case, be aware of the following risks if you must shut down the secondary system for any length of time:
 - The local trail on the primary system could run out of disk space as data accumulates while the standby is offline. This will cause the primary Extract to abend.
 - If the primary system fails while the standby is offline, the data changes will not be available to be applied to the live standby when it is functional again, thereby breaking the synchronized state and requiring a full re-instantiation of the live standby.
2. On the **primary** system, stop the user applications, but leave the primary Extract and the data pump on that system running so that they capture any backlogged transaction data.
3. On the **primary** system, issue the following command for the primary Extract until it returns "At EOF, no more records to process." This indicates that all transactions are now captured.

```
LAG EXTRACT ext_1
```

Note:

Since capture continues to read REDO, the non-production workload continues to work. In this case, there is possibility that At EOF is never returned even though the production workload has already stopped8.5.1..

4. On the **primary** system, stop the primary Extract process

```
STOP EXTRACT ext_1
```

5. On the **primary** system, issue the following command for the data pump until it returns "At EOF, no more records to process." This indicates that the pump sent all of the captured data to the live standby.

```
LAG EXTRACT pump_1
```

6. On the **primary** system, stop the data pump.

```
STOP EXTRACT pump_1
```

7. On the **live standby** system, issue the `STATUS REPLICAT` command until it returns "At EOF (end of file)." This confirms that Replicat applied all of the data from the trail to the database.

```
STATUS REPLICAT rep_1
```

8. On the **live standby** system, stop Replicat.

```
STOP REPLICAT rep_1
```

9. On the **live standby** system, do the following:

- Run the script that grants insert, update, and delete permissions to the users of the business applications.
- Run the script that enables triggers and cascade delete constraints.
- Run the scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.

10. On the **live standby** system, alter the primary Extract to begin capturing data based on the current timestamp. Otherwise, Extract will spend unnecessary time looking for operations that date back to the time that the group was created with the `ADD EXTRACT` command.

```
ALTER EXTRACT ext_2, BEGIN NOW
```

11. On the **live standby** system, start the primary Extract so that it is ready to capture transactional changes.

```
START EXTRACT ext_2
```

 **Note:**

Do not start the data pump on the **live standby** system, and do not start the Replicat on the **primary** system. Data must be stored in the local trail on the live standby until the primary database is ready for user activity again.

12. Switch user activity to the **live standby** system.
13. On the **primary system**, perform the system maintenance.

Moving User Activity Back to the Primary System

To move user activity back to the primary system:

1. On the **live standby** system, stop the user applications, but leave the primary Extract running so that it captures any backlogged transaction data.
2. On the **primary** system, start Replicat in preparation to receive changes from the live standby system.

```
START REPLICAT rep_2
```

3. On the **live standby** system, start the data pump to begin moving the data that is stored in the local trail across TCP/IP to the primary system.

```
START EXTRACT pump_2
```

4. On the **live standby** system, issue the following command for the primary Extract until it returns "At EOF, no more records to process." This indicates that all transactions are now captured.

```
LAG EXTRACT ext_2
```

5. On the **live standby** system, stop the primary Extract.

```
STOP EXTRACT ext_2
```

6. On the **live standby** system, issue the following command for the data pump until it returns "At EOF, no more records to process." This indicates that the pump sent all of the captured data to the primary system.

```
LAG EXTRACT pump_2
```

7. On the **live standby** system, stop the data pump.

```
STOP EXTRACT pump_2
```

8. On the **primary** system, issue the `STATUS REPLICAT` command until it returns "At EOF (end of file)." This confirms that Replicat applied all of the data from the trail to the database.

```
STATUS REPLICAT rep_2
```

9. On the **primary** system, stop Replicat.

```
STOP REPLICAT rep_2
```

10. On the **primary** system, do the following:

- Run the script that grants insert, update, and delete permissions to the users of the business applications.
- Run the script that enables triggers and cascade delete constraints.
- Run the scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.

11. On the **primary** system, alter the primary Extract to begin capturing data based on the current timestamp. Otherwise, Extract will spend unnecessary time looking for operations that were already captured and replicated while users were working on the standby system.

```
ALTER EXTRACT ext_1, BEGIN NOW
```

12. On the **primary** system, start the primary Extract so that it is ready to capture transactional changes.

```
START EXTRACT ext_1
```

13. Switch user activity to the **primary** system.

14. (Optional) If system maintenance must be done on the **live standby** system, you can do it now, before starting the data pump on the primary system. Note that captured data will be accumulating on the primary system while the standby is offline.

15. On the **primary** system, start the data pump.

```
START EXTRACT pump_1
```

16. On the **live standby** system, start Replicat.

```
START REPLICAT rep_1
```

Moving User Activity in an Unplanned Failover

These sections describe how to move user activity in an unplanned failover.

- [Moving User Activity to the Live Standby](#)
- [Moving User Activity Back to the Primary System](#)

Moving User Activity to the Live Standby

This procedure does the following:

- Prepares the live standby for user activity.
- Ensures that all transactions from the primary system are applied to the live standby.
- Activates Oracle GoldenGate to capture transactional changes on the live standby.
- Moves users to the live standby system.

Perform these steps on the live standby system

To move users to the live standby

1. Issue the `STATUS REPLICAT` command until it returns "At EOF (end of file)" to confirm that Replicat applied all of the data from the trail to the database.

```
STATUS REPLICAT rep_1
```

2. Stop the Replicat process.

```
STOP REPLICAT rep_1
```

3. Run the script that grants insert, update, and delete permissions to the users of the business applications.
4. Run the script that enables triggers and cascade delete constraints.
5. Run the scripts that fail over the application server, start applications, and copy essential files that are not part of the replication environment.
6. Start the primary Extract process on the live standby.

```
START EXTRACT ext_2
```

7. Move the users to the standby system and let them start working.

Note:

Do not start the data pump group on the standby. The user transactions must accumulate there until just before user activity is moved back to the primary system.

Moving User Activity Back to the Primary System

This procedure does the following:

- Recovers the Oracle GoldenGate environment.
- Makes a copy of the live standby data to the restored primary system.
- Propagates user transactions that occurred while the copy was being made.
- Reconciles the results of the copy with the propagated changes.
- Moves users from the standby system to the restored primary system.
- Prepares replication to maintain the live standby again.

Perform these steps after the recovery of the primary system is complete.

To Recover the Source Oracle GoldenGate Environment

1. On the **primary** system, recover the Oracle GoldenGate directory from your backups.
2. On the **primary** system, run GGSCI.
3. On the **primary** system, delete the primary Extract group.

```
DELETE EXTRACT ext_1
```

4. On the **primary** system, delete the local trail.

```
DELETE EXTTRAIL local_trail_1
```

5. On the **primary** system, add the primary Extract group again, using the same name so that it matches the parameter file that you restored from backup. For documentation purposes, this group is called *ext_1*. This step initializes the Extract checkpoint from its state before the failure to a clean state.

```
ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time  
[, THREADS n]
```

- For TRANLOG and INTEGRATED TRANLOG, see *Reference for Oracle GoldenGate*. INTEGRATED TRANLOG enables integrated capture for an Oracle database.

6. On the **primary** system, add the local trail again, using the same name as before. For documentation purposes, this trail is called *local_trail_1*.

```
ADD EXTTRAIL local_trail_1, EXTRACT ext_1
```

- For EXTRACT, specify the *ext_1* group to write to this trail.

7. On the **primary** system, start the Manager process.

```
START MANAGER
```

To Copy the Database from Standby to Primary System

1. On the **primary** system, run scripts to disable triggers and cascade delete constraints.
2. On the **standby** system, start making a hot copy of the database.
3. On the **standby** system, record the time at which the copy finishes.
4. On the **standby system**, stop user access to the applications. Allow all open transactions to be completed.

To Propagate Data Changes Made During the Copy

1. On the **primary** system, start Replicat.

```
START REPLICAT rep_2
```

2. On the **live standby** system, start the data pump. This begins transmission of the accumulated user transactions from the standby to the trail on the primary system.

```
START EXTRACT pump_2
```

3. On the **primary** system, issue the `INFO REPLICAT` command until you see that it posted all of the data changes that users generated on the standby system during the initial load. Refer to the time that you recorded previously. For example, if the copy stopped at 12:05, make sure that change replication has posted data up to that point.

```
INFO REPLICAT rep_2
```

4. On the **primary** system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

```
SEND REPLICAT rep_2, NOHANDLECOLLISIONS
```

5. On the **primary** system, issue the `STATUS REPLICAT` command until it returns "At EOF (end of file)" to confirm that Replicat applied all of the data from the trail to the database.

```
STATUS REPLICAT rep_2
```

6. On the **live standby** system, stop the data pump. This stops transmission of any user transactions from the standby to the trail on the primary system.

```
STOP EXTRACT pump_2
```

7. On the **primary** system, stop the Replicat process.

```
STOP REPLICAT rep_2
```

At this point in time, the primary and standby databases should be in a state of synchronization again.

(Optional) To Verify Synchronization

1. Use a compare tool, such as Oracle GoldenGate Veridata, to compare the source and standby databases for parity.
2. Use a repair tool, such as Oracle GoldenGate Veridata, to repair any out-of-sync conditions.

To Switch Users to the Primary System

1. On the **primary** system, run the script that grants insert, update, and delete permissions to the users of the business applications.
2. On the **primary** system, run the script that enables triggers and cascade delete constraints.
3. On the **primary** system, run the scripts that fail over the application server, start applications, and copy essential files that are not part of the replication environment.
4. On the **primary** system, start the primary Extract process.

```
START EXTRACT ext_1
```

5. On the **primary** system, allow users to access the applications.

9

Configuring Oracle GoldenGate for Active-Active High Availability

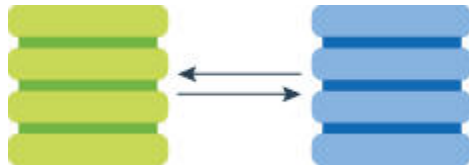
This chapter describes how to configure Oracle GoldenGate for active-active high availability.

Topics:

- [Overview of an Active-Active Configuration](#)
- [Considerations for an Active-Active Configuration](#)
- [Preventing Data Looping](#)
- [Managing Conflicts](#)
- [Additional Information](#)
- [Creating an Active-Active Configuration](#)

Overview of an Active-Active Configuration

Oracle GoldenGate supports an active-active bi-directional configuration, where there are two systems with identical sets of data that can be changed by application users on either system. Oracle GoldenGate replicates transactional data changes from each database to the other to keep both sets of data current.



In a bi-directional configuration, there is a complete set of active Oracle GoldenGate processes on each system. Data captured by an Extract process on one system is propagated to the other system, where it is applied by a local Replicat process.

This configuration supports load sharing. It can be used for disaster tolerance if the business applications are identical on any two peers. Bi-directional synchronization is supported for all database types that are supported by Oracle GoldenGate.

Oracle GoldenGate supports active-active configurations for:

- DB2 on z/OS, LUW, and IBM i
- MySQL
- Oracle
- SQL Server

Oracle GoldenGate supports DDL replication in an Oracle active-active configuration. DDL support is available for Oracle Database and MySQL (like-to-like configuration) databases.

Considerations for an Active-Active Configuration

The following considerations apply in an active-active configuration. In addition, review the Oracle GoldenGate installation and configuration document for your type of database to see if there are any other limitations or requirements to support a bi-directional configuration.

- [TRUNCATES](#)
- [Application Design](#)
- [Keys](#)
- [Triggers and Cascaded Deletes](#)
- [Database-Generated Values](#)
- [Database Configuration](#)

TRUNCATES

Bi-directional replication of `TRUNCATES` is not supported, but you can configure these operations to be replicated in one direction, while data is replicated in both directions. To replicate `TRUNCATES` (if supported by Oracle GoldenGate for the database) in an active-active configuration, the `TRUNCATES` must originate only from one database, and only from the same database each time.

Configure the environment as follows:

- Configure all database roles so that they cannot execute `TRUNCATE` from any database other than the one that is designated for this purpose.
- On the system where `TRUNCATE` will be permitted, configure the Extract and Replicat parameter files to contain the `GETTRUNCATES` parameter.
- On the other system, configure the Extract and Replicat parameter files to contain the `IGNORETRUNCATES` parameter. No `TRUNCATES` should be performed on this system by applications that are part of the Oracle GoldenGate configuration.

Application Design

When using Active-Active replication, the time zones must be the same on both systems so that timestamp-based conflict resolution and detection can operate.

Active-active replication is not recommended for use with commercially available packaged business applications, unless the application is designed to support it. Among the obstacles that these applications present are:

- Packaged applications might contain objects and data types that are not supported by Oracle GoldenGate.
- They might perform automatic DML operations that you cannot control, but which will be replicated by Oracle GoldenGate and cause conflicts when applied by Replicat.

- You probably cannot control the data structures to make modifications that are required for active-active replication.

Keys

For accurate detection of conflicts, all records must have a unique, not-null identifier. If possible, create a primary key. If that is not possible, use a unique key or create a substitute key with a `KEYCOLS` option of the `MAP` and `TABLE` parameters. In the absence of a unique identifier, Oracle GoldenGate uses all of the columns that are valid in a `WHERE` clause, but this will degrade performance if the table contains numerous columns.

To maintain data integrity and prevent errors, the following must be true of the key that you use for any given table:

- contain the same columns in all of the databases where that table resides.
- contain the same values in each set of corresponding rows across the databases.

Triggers and Cascaded Deletes

Triggers and `ON DELETE CASCADE` constraints generate DML operations that can be replicated by Oracle GoldenGate. To prevent the local DML from conflicting with the replicated DML from these operations, do the following:

- Modify triggers to ignore DML operations that are applied by Replicat. If the target is an Oracle database, Replicat handles triggers without any additional configuration when in integrated mode. Parameter options are available for a nonintegrated Replicat for Oracle. See *Diabling Triggers and Referential Cascade Constraints on Target Tables in Using Oracle GoldenGate for Oracle Database*.
- Disable `ON DELETE CASCADE` constraints and use a trigger on the parent table to perform the required delete(s) to the child tables. Create it as a `BEFORE` trigger so that the child tables are deleted before the delete operation is performed on the parent table. This reverses the logical order of a cascaded delete but is necessary so that the operations are replicated in the correct order to prevent "table not found" errors on the target.



Note:

For MySQL targets, cascade delete queries result in the deletion of the child of the parent operation.



Note:

For Oracle Database targets, if Replicat is in integrated mode, constraints are handled automatically without special configuration.

Database-Generated Values

Do not replicate database-generated sequential values, such as Oracle sequences, in a bi-directional configuration. The range of values must be different on each system, with no chance of overlap. For example, in a two-database environment, you can have one server generate even values, and the other odd. For an n -server environment, start each key at a different value and increment the values by the number of servers in the environment. This method may not be available to all types of applications or databases. If the application permits, you can add a location identifier to the value to enforce uniqueness.

Database Configuration

One of the databases must be designated as the *trusted source*. This is the primary database and its host system from which the other database is derived in the initial synchronization phase and in any subsequent resynchronizations that become necessary. Maintain frequent backups of the trusted source data.

Preventing Data Looping

In a bidirectional configuration, SQL changes that are replicated from one system to another must be prevented from being replicated back to the first system. Otherwise, it moves back and forth in an endless loop, as in this example:

1. A user application updates a row on system A.
2. Extract extracts the row on system A and sends it to system B.
3. Replicat updates the row on system B.
4. Extract extracts the row on system B and sends it back to system A.
5. The row is applied on system A (for the second time).
6. This loop continues endlessly.

To prevent data loopback, you may need to provide instructions that:

- prevent the capture of SQL operations that are generated by Replicat, but enable the capture of SQL operations that are generated by business applications if they contain objects that are specified in the Extract parameter file.
- identify local Replicat transactions, in order for the Extract process to ignore them.
- [Preventing the Capture of Replicat Operations](#)
- [Identifying Replicat Transactions](#)
- [Replicating DDL in a Bi-directional Configuration](#)

Preventing the Capture of Replicat Operations

Depending on which database you are using, you may or may not need to provide explicit instructions to prevent the capture of Replicat operations.

- [Preventing the Capture of Replicat Transactions \(Oracle\)](#)
- [Preventing Capture of Replicat Transactions \(Other Databases\)](#)

Preventing the Capture of Replicat Transactions (Oracle)

To prevent the capture of SQL that is applied by Replicat to an Oracle database, there are different options depending on the Extract capture mode:

- When Extract is in classic or integrated capture mode, use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG tag` option. This parameter directs the Extract process to ignore transactions that are tagged with the specified redo tag. See [Identifying Replicat Transactions](#) to set the tag value.
- When Extract is in classic capture mode, use the Extract `TRANLOGOPTIONS` parameter with the `EXCLUDEUSER` or `EXCLUDEUSERID` option to exclude the user name or ID that is used by Replicat to apply the DDL and DML transactions. Multiple `EXCLUDEUSER` statements can be used. The specified user is subject to the rules of the `GETREPLICATES` or `IGNOREREPLICATES` parameter. See [Preventing Capture of Replicat Transactions \(Other Databases\)](#) for more information.

Preventing Capture of Replicat Transactions (Other Databases)

To prevent the capture of SQL that is applied by Replicat to other database types (including Oracle, if Extract operates in classic capture mode), use the following parameters:

- `GETAPPLOPS` | `IGNOREAPPLOPS`: Controls whether or not data operations (DML) produced by business applications *except Replicat* are included in the content that Extract writes to a specific trail or file.
- `GETREPLICATES` | `IGNOREREPLICATES`: Controls whether or not DML operations produced by *Replicat* are included in the content that Extract writes to a specific trail or file.

Identifying Replicat Transactions

To configure Extract to identify Replicat transactions, follow the instructions for the database from which Extract will capture data.

- [DB2 z/OS, DB2 LUW, and DB2 for i](#)
- [MySQL](#)
- [Oracle](#)
- [SQL Server](#)

DB2 z/OS, DB2 LUW, and DB2 for i

Identify the Replicat user name by using the following parameter statement in the Extract parameter file.

```
TRANLOGOPTIONS EXCLUDEUSER user
```

This parameter statement marks all DDL and DML transactions that are generated by this user as Replicat transactions. The user name is included in the transaction record that is read by Extract.

MySQL

Identify the name of the Replicat checkpoint table by using the following parameter statement in the Extract parameter file.

```
TRANLOGOPTIONS FILTERTABLE table_name
```

Replicat writes a checkpoint to the checkpoint table at the end of each of its transactions as part of its checkpoint procedure. (This is the table that is created with the `ADD CHECKPOINTTABLE` command.) Because every Replicat transaction includes a write to this table, it can be used to identify Replicat transactions in a bidirectional configuration. `FILTERTABLE` identifies the name of the checkpoint table, so that Extract ignores transactions that contain any operations on it.

Oracle

There are multiple ways to identify Replicat transaction in an Oracle environment. When Replicat is in classic or integrated mode, you use the following parameters:

- Use `DBOPTIONS` with the `SETTAG` option in the Replicat parameter file. Replicat tags the transactions being applied with the specified value, which identifies those transactions in the redo stream. The default `SETTAG` value is `00`. Valid values are a single `TAG` value consisting of hexadecimal digits.
- Use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG` option in the Extract parameter file. The logmining server associated with that Extract excludes redo that is tagged with the `SETTAG` value.

The following shows how `SETTAG` can be set in the Replicat parameter file:

```
DBOPTIONS SETTAG 0935
```

The following shows how `EXCLUDETAG` can be set in the Extract parameter file:

```
TRANLOGOPTIONS EXCLUDETAG 0935
```

If you are excluding multiple tags, each must have a separate `TRANLOGOPTIONS EXCLUDETAG` statement specified.

You can also use the transaction name or userid of the Replicat user to identify Replicat transactions. You can choose which of these to ignore when you configure Extract. See [Preventing the Capture of Replicat Transactions \(Oracle\)](#).

For more information, see *Reference for Oracle GoldenGate*.

SQL Server

(CDC Extract) Identify the name of the Replicat checkpoint table by using the following parameter statement in the Extract parameter file and ensure that the Replicat checkpoint table has been enabled for supplemental logging with the `ADD TRANDATA` command.

```
TRANLOGOPTIONS FILTERTABLE table_name
```

Replicat writes a checkpoint to the checkpoint table at the end of each of its transactions as part of its checkpoint procedure. (This is the table that is created with the `ADD CHECKPOINTTABLE` command). Because every Replicat transaction includes a

write to this table, it can be used to identify Replicat transactions in a bi-directional configuration. `FILTERTABLE` identifies the name of the checkpoint table, so that Extract ignores transactions that contain any operations on it.

(Classic Extract) By default, Extract ignores the Replicat's transactions, however, if you modify the Replicat's transaction name with the `DBOPTIONS TRANSNAME` parameter, then you must exclude those transactions by using the following parameter statement in the Extract parameter file.

```
TRANLOGOPTIONS EXCLUDETRANS transaction_name
```

This parameter statement is only required if the Replicat transaction name is set to something other than the default of `ggs_repl`.

Replicating DDL in a Bi-directional Configuration

Additional consideration must be taken when replicating DDL bi-directionally, currently only supported for Oracle database. For more information, see *Managing the DDL Replication Environment Using Oracle GoldenGate for Oracle Database*.

Managing Conflicts

Uniform conflict-resolution procedures must be in place on all systems in an active-active configuration. Conflicts should be identified immediately and handled with as much automation as possible; however, different business applications will present their own unique set of requirements in this area.

Because Oracle GoldenGate is an asynchronous solution, conflicts can occur when modifications are made to identical sets of data on separate systems at (or almost at) the same time. Conflicts occur when the timing of simultaneous changes results in one of these out-of-sync conditions:

- A **uniqueness conflict** occurs when Replicat applies an insert or update operation that violates a uniqueness integrity constraint, such as a `PRIMARY KEY` or `UNIQUE` constraint. An example of this conflict type is when two transactions originate from two different databases, and each one inserts a row into a table with the same primary key value.
- An **update conflict** occurs when Replicat applies an update that conflicts with another update to the same row. Update conflicts happen when two transactions that originate from different databases update the same row at nearly the same time. Replicat detects an update conflict when there is a difference between the old values (the before values) that are stored in the trail record and the current values of the same row in the target database.
- A **delete conflict** occurs when two transactions originate at different databases, and one deletes a row while the other updates or deletes the same row. In this case, the row does not exist to be either updated or deleted. Replicat cannot find the row because the primary key does not exist.

For example, UserA on DatabaseA updates a row, and UserB on DatabaseB updates the same row. If UserB's transaction occurs before UserA's transaction is synchronized to DatabaseB, there will be a conflict on the replicated transaction.

A more complicated example involves three databases and illustrates a more complex ordering conflict. Assume three databases A, B, and C. Suppose a user inserts a row

at database A, which is then replicated to database B. Another user then modifies the row at database B, and the row modification is replicated to database C. If the row modification from B arrives at database C before the row insert from database A, C will detect a conflict.

Where possible, try to minimize or eliminate any chance of conflict. Some ways to do so are:

- Configure the applications to restrict which columns can be modified in each database. For example, you could limit access based on geographical area, such as by allowing different sales regions to modify only the records of their own customers. As another example, you could allow a customer service application on one database to modify only the `NAME` and `ADDRESS` columns of a customer table, while allowing a financial application on another database to modify only the `BALANCE` column. In each of those cases, there cannot be a conflict caused by concurrent updates to the same record.
- Keep synchronization latency low. If UserA on DatabaseA and UserB on DatabaseB both update the same rows at about the same time, and UserA's transaction gets replicated to the target row before UserB's transaction is completed, conflict is avoided. See [Tuning the Performance of Oracle GoldenGate](#) for suggestions on improving the performance of the Oracle GoldenGate processes.

To avoid conflicts, replication latency must be kept as low as possible. When conflicts are unavoidable, they must be identified immediately and resolved with as much automation as possible, either through the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature, or through methods developed on your own. Custom methods can be integrated into Oracle GoldenGate processing through the `SQLEXEC` and user exit functionality. See [Configuring Conflict Detection and Resolution](#) for more information about using Oracle GoldenGate to handle conflicts.

For Oracle database, the automatic Conflict Detection Resolution (CDR) feature exists. To know more, see Oracle GoldenGate Automatic Conflict Detection and Resolution in the *Oracle Database XStream Guide*.

Additional Information

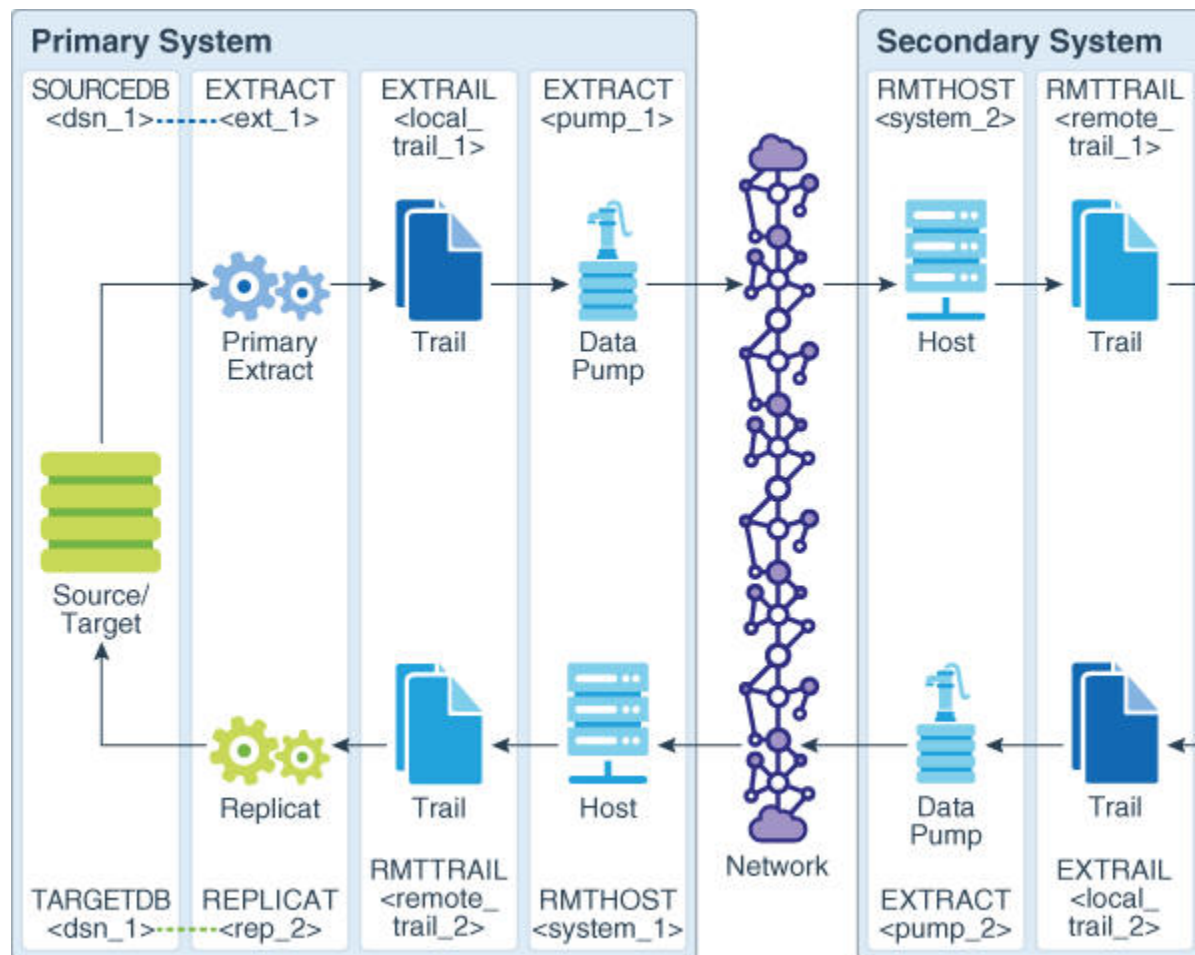
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see the Oracle GoldenGate installation and configuration document for your database type.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see [Configuring Online Change Synchronization](#).
- For additional tuning options for Oracle GoldenGate, see [Tuning the Performance of Oracle GoldenGate](#).
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see *Reference for Oracle GoldenGate*.

Creating an Active-Active Configuration

Refer to [Figure 9-1](#) for a visual representation of the objects you will be creating.

Figure 9-1 Oracle GoldenGate Configuration for Active-active Synchronization



- Prerequisites on Both Systems
- Configuration from Primary System to Secondary System
- Configuration from Secondary System to Primary System

Prerequisites on Both Systems

Perform these prerequisite tasks on both systems:

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). See [Creating a Checkpoint Table](#) for instructions.
2. Configure the Manager process. See [Configuring Manager and Network Communications](#) for instructions.

Configuration from Primary System to Secondary System

These steps add the processes necessary to send data from the primary system to the secondary database.

To Configure the Primary Extract Group

Perform these steps on the primary system.

1. Use the `ADD EXTRACT` command to create a primary Extract group. For documentation purposes, this group is called `ext_1`.

```
ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
```

2. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called `local_trail_1`.

```
ADD EXTTRAIL local_trail_1, EXTRACT ext_1
```

For `EXTRACT`, specify the `ext_1` group to write to this trail

3. Use the `EDIT PARAMS` command to create a parameter file for the `ext_1` group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_1
-- Exclude Replicat transactions. Uncomment ONE of the following:
-- DB2 z/OS, DB2 LUW, DB2 IBM i, and Oracle (classic capture):
-- TRANLOGOPTIONS EXCLUDEUSER Replicat_user
-- Oracle (classic capture) alternative to EXCLUDEUSER:
-- EXCLUDEUSERID Oracle_uid
-- Oracle integrated capture:
-- EXCLUDETAG tag
-- SQL Server:
-- TRANLOGOPTIONS EXCLUDETRANS transaction_name
-- -- Teradata:
-- SQLEXEC 'SET SESSION OVERRIDE REPLICATION ON;'
-- SQLEXEC 'COMMIT;'
-- Specify API commands if Teradata:
VAM library name, PARAMS ('param' [, 'param'] [, ...])
-- Capture before images for conflict resolution:
GETBEFORECOLS (ON operation {ALL | KEY | KEYINCLUDING (col_list) |
ALLEXCLUDING (col_list)})
-- Log all scheduling columns for CDR and if using integrated Replicat
LOGALLSUPCOLS
-- Specify tables to be captured and (optional) columns to fetch:
TABLE [container.[catalog.]owner.* [, FETCHCOLS cols | FETCHCOLSEXCEPT cols];
```

 **Note:**

The VAM parameter in the examples is used only for heterogeneous databases and does not apply to Oracle Database.

To Configure the Data Pump

Perform these steps on the primary system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called `pump_1`.

```
ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
```

For `EXTTRAILSOURCE`, specify `local_trail_1` as the data source.

2. Use the `ADD RMTTRAIL` command to add a remote trail that will be created on the secondary system. For documentation purposes, this trail is called `remote_trail_1`.

```
ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
```

For `EXTRACT`, specify the `pump_1` data pump to write to this trail.

See *Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

Use the `EDIT PARAMS` command to create a parameter file for the `pump_1` group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the secondary system
-- and optional encryption of data over TCP/IP:
RMTTHOSTOPTIONS system_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on secondary system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables to be captured:
TABLE [container.|catalog.]owner.*;
```

To Configure the Replicat Group

Perform these steps on the secondary system.

1. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called `rep_1`.

```
ADD REPLICAT rep_1
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_1, BEGIN time
```

For `EXTTRAIL`, specify `remote_trail_1` as the trail that this Replicat reads.

2. Use the `EDIT PARAMS` command to create a parameter file for the `rep_1` group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep_1
```

```

-- State that source and target definitions are identical:
ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Set redo tag for Oracle only replicat via settag
-- Default is 00.
SETTAG tag_value
-- Specify tables for delivery, threads if coordinated Replicat
-- and conflict-resolution:
MAP [container.catalog.owner.*, TARGET owner.*, COMPARECOLS (ON operation
{ALL | KEY | KEYINCLUDING (col_list) | ALLEXCLUDING (col_list)}),
RESOLVECONFLICT (conflict type (resolution_name, resolution_type COLS
(col[,...]))
[, THREAD (thread_ID)]
[, THREADRANGE (thread_range[, column_list])]
;
-- Specify mapping of exceptions to exceptions table:
MAP [container.catalog.owner.*, TARGET owner.exceptions, EXCEPTIONSONLY;

```

Configuration from Secondary System to Primary System

These steps add the processes necessary to send data from the secondary system to the primary database.

To Configure the Primary Extract Group

Perform these steps on the secondary system.



Note:

This is a reverse image of the configuration that you just created.

1. Use the ADD EXTRACT command to create a primary Extract group. For documentation purposes, this group is called *ext_2*.

```
ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
```

2. Use the ADD EXTTRAIL command to add a local trail. For documentation purposes, this trail is called *local_trail_2*.

```
ADD EXTTRAIL local_trail_2, EXTRACT ext_2
```

For EXTRACT, specify the *ext_2* group to write to this trail.

3. Use the EDIT PARAMS command to create a parameter file for the *ext_2* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```

-- Identify the Extract group:
EXTRACT ext_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Specify the local trail that this Extract writes to

```

```

-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_2
-- Exclude Replicat transactions. Uncomment ONE of the following:
-- DB2 z/OS, DB2 LUW, DB2 IBM i, and Oracle:
-- TRANLOGOPTIONS EXCLUDEUSER Replicat_user
-- Oracle alternative to EXCLUDEUSER:
-- EXCLUDEUSERID Oracle_uid
-- Oracle integrated capture:
-- EXCLUDETAG tag
-- SQL Server:
-- TRANLOGOPTIONS EXCLUDETRANS transaction_name
---- Teradata:
-- SQLEXEC 'SET SESSION OVERRIDE REPLICATION ON;'
-- SQLEXEC 'COMMIT;'
-- Oracle:
-- TRACETABLE trace_table_name
-- Log all scheduling columns for CDR and if using integrated Replicat
LOGALLSUPCOLS
-- Capture before images for conflict resolution:
GETBEFORECOLS (ON operation {ALL | KEY | KEYINCLUDING (col_list) |
ALLEXCLUDING (col_list)})
-- Specify tables to be captured and (optional) columns to fetch:
TABLE [container.|catalog.]owner.* [, FETCHCOLS cols | FETCHCOLSEXCEPT cols];

```

 **Note:**

To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node. For more information on identifying these tables and configuring DBFS for propagation by Oracle GoldenGate, see *Applying the Required Patch in Using Oracle GoldenGate for Oracle Database*.

To Configure the Data Pump

Perform these steps on the secondary system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called `pump_2`.

```
ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
```

For `EXTTRAILSOURCE`, specify `local_trail_2` as the data source.

2. Use the `ADD RMTTRAIL` command to add a remote trail that will be created on the primary system. For documentation purposes, this trail is called `remote_trail_2`.

```
ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
```

For `EXTRACT`, specify the `pump_2` data pump to write to this trail.

3. Use the `EDIT PARAMS` command to create a parameter file for the `pump_2` group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_2
```

```
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the primary system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS system_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the primary system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables to be captured:
TABLE [container.|catalog.]owner.*;
```

 **Note:**

To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node. For more information on identifying these tables and configuring DBFS for propagation by Oracle GoldenGate, see *Configuring the DBFS File System* in *Using Oracle GoldenGate for Oracle Database*.

To Configure the Replicat Group

Perform these steps on the primary system.

1. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called `rep_2`.

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2, BEGIN time
```

For `EXTTRAIL`, specify `remote_trail_2` as the trail that this Replicat reads.

See `ADD RMTTRAIL` in *Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

2. Use the `EDIT PARAMS` command to create a parameter file for the `rep_2` group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the Replicat group:
REPLICAT rep_2
-- State that source and target definitions are identical:
ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB dsn_1][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery, threads if coordinated Replicat
-- and conflict-resolution:
MAP [container.|catalog.]owner.*, TARGET owner.*, COMPARECOLS (ON operation
{ALL | KEY | KEYINCLUDING (col_list) | ALLEXCLUDING (col_list)}),
RESOLVECONFLICT (conflict type (resolution_name, resolution_type COLS
(col[,...]))
[, THREAD (thread_ID)]
[, THREADRANGE (thread_range[, column_list])])
```

```
;  
-- Specify mapping of exceptions to exceptions table:  
MAP [container.|catalog.]owner.*, TARGET owner.exceptions, EXCEPTIONSONLY;
```

 **Note:**

To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node.

10

Configuring Conflict Detection and Resolution

This chapter contains instructions for using the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature. Conflict detection and resolution is required in active-active configurations, where Oracle GoldenGate must maintain data synchronization among multiple databases that contain the same data sets.

Topics:

- [Overview of the Oracle GoldenGate CDR Feature](#)
- [Configuring Oracle GoldenGate CDR](#)
- [CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD](#)
- [CDR Example 2: UPDATEROWEXISTS with USEDELTA and USEMAX](#)
- [CDR Example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE](#)

Overview of the Oracle GoldenGate CDR Feature

Oracle GoldenGate Conflict Detection and Resolution (CDR) provides basic conflict resolution routines that:

- Resolve a uniqueness conflict for an `INSERT`.
- Resolve a "no data found" conflict for an `UPDATE` when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a "no data found" conflict for an `UPDATE` when the row does not exist.
- Resolve a "no data found" conflict for a `DELETE` when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a "no data found" conflict for a `DELETE` when the row does not exist.

To use conflict detection and resolution (CDR), the target database must reside on a Windows, Linux, or UNIX system. It is not supported for databases on the NonStop platform.

CDR supports scalar data types such as:

- `NUMERIC`
- `DATE`
- `TIMESTAMP`
- `CHAR/NCHAR`
- `VARCHAR/ NVARCHAR`

This means that these column types can be used with the `COMPARECOLS` parameter, the `GETBEFORECOLS` parameter, and as the resolution column in the `USEMIN` and `USEMAX`

options of the `RESOLVECONFLICT` parameter. Only `NUMERIC` columns can be used for the `USEDELTA` option of `RESOLVECONFLICT`. Do not use CDR for columns that contain LOBs, abstract data types (ADT), or user-defined types (UDT).

Conflict resolution is not performed when Replicat operates in `BATCHSQL` mode. If a conflict occurs in `BATCHSQL` mode, Replicat reverts to `GROUPTRANSOPS` mode, and then to single-transaction mode. Conflict detection occurs in all three modes. For more information, see *Reference for Oracle GoldenGate*.

Configuring Oracle GoldenGate CDR

Here are the steps to configure the source database, target database, and Oracle GoldenGate for conflict detection and resolution.

Topics:

- [Making the Required Column Values Available to Extract](#)
- [Configuring the Oracle GoldenGate Parameter Files for Conflict Resolution](#)
- [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#)
- [Viewing CDR Statistics](#)

Making the Required Column Values Available to Extract

To use CDR, the following column values must be logged so that Extract can write them to the trail.

- The full before image of each record. Some databases do not provide a before image in the log record, and must be configured to do so with supplemental logging. For most supported databases, you can use the `ADD TRANDATA` command for this purpose.
- Use the `LOGALLSUPCOLS` parameter to ensure that the full before and after images of the scheduling columns are written to the trail. Scheduling columns are primary key, unique index, and foreign key columns. `LOGALLSUPCOLS` causes Extract to include in the trail record the before image for `UPDATE` operations and the before image of all supplementally logged columns for both `UPDATE` and `DELETE` operations.

For detailed information about these parameters and commands, see the *Reference for Oracle GoldenGate*. See the examples starting on [CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD](#) for more information on how these parameters work with CDR.

Configuring the Oracle GoldenGate Parameter Files for Conflict Resolution

The following parameters are required to support conflict detection and resolution.

1. Use the `GETBEFORECOLS` option of the Extract `TABLE` parameter to specify columns for which you want Extract to capture the before image of an update or delete operation. For DB2 databases, use the `GETUPDATEBEFORES` parameter instead of `GETBEFORECOLS`, which is not supported for DB2.

2. Use the `COMPARECOLS` option of the `MAP` parameter in the Replicat parameter file to specify columns that are to be used with before values in the Replicat `WHERE` clause. The before values are compared with the current values in the target database to detect update and delete conflicts. (By default, Replicat only uses the primary key in the `WHERE` clause; this may not be enough for conflict detection).
3. Use the `RESOLVECONFLICT` option of the `MAP` parameter to specify conflict resolution routines for different operations and conflict types. You can use `RESOLVECONFLICT` multiple times in a `MAP` statement to specify different resolutions for different conflict types. However, you cannot use `RESOLVECONFLICT` multiple times for the same type of conflict. Use identical conflict-resolution procedures on all databases, so that the same conflict produces the same end result. One conflict-resolution method might not work for every conflict that could occur. You might need to create several routines that can be called in a logical order of priority so that the risk of failure is minimized.

 **Note:**

Additional consideration should be given when a table has a primary key and additional unique indexes or unique keys. The automated routines provided with the `COMPARECOLS` and `RESOLVECONFLICT` parameters require a consistent way to uniquely identify each row. Failure to consistently identify a row will result in an error during conflict resolution. In these situations the additional unique keys should be disabled or you can use the `SQLEXEC` feature to handle the error thrown and resolve the conflict.

For detailed information about these parameters, see *Reference for Oracle GoldenGate*. See the examples starting on [CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD](#), for more information on these parameters.

Configuring the Oracle GoldenGate Parameter Files for Error Handling

CDR should be used in conjunction with error handling to capture errors that were resolved and errors that CDR could not resolve.

1. Conflict resolution is performed before these other error-handling parameters: `HANDLECOLLISIONS`, `INSERTMISSINGUPDATES`, and `REPERROR`. Use the `REPERROR` parameter to assign rules for handling errors that cannot be resolved by CDR, or for errors that you do not want to handle through CDR. It might be appropriate to have `REPERROR` handle some errors, and CDR handle others; however, if `REPERROR` and CDR are configured to handle the same conflict, CDR takes precedence. The `INSERTMISSINGUPDATES` and `HANDLECOLLISIONS` parameters also can be used to handle some errors not handled by CDR. See the *Reference for Oracle GoldenGate* for details about these parameters.
2. (Optional) Create an exceptions table. When an exceptions table is used with an exceptions `MAP` statment (see [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#)), Replicat sends every operation that generates a conflict (resolved or not) to the exceptions `MAP` statement to be mapped to the exceptions table. Omit a primary key on this table if Replicat is to process `UPDATE` and `DELETE` conflicts; otherwise there can be integrity constraint errors.

At minimum, an exceptions table should contain the same columns as the target table. These rows will contain each row image that Replicat applied to the target (or tried to apply).

In addition, you can define additional columns to capture other information that helps put the data in transactional context. Oracle GoldenGate provides tools to capture this information through the exceptions `MAP` statement (see [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#)). Such columns can be, but are not limited to, the following:

- The before image of the trail record. This is a duplicate set of the target columns with names such as `col1_before`, `col2_before`, and so forth.
 - The current values of the target columns. This also is a duplicate set of the target columns with names such as `col1_current`, `col2_current`, and so forth.
 - The name of the target table
 - The timestamp of the conflict
 - The operation type
 - The database error number
 - (Optional) The database error message
 - Whether the conflict was resolved or not
3. Create an exceptions `MAP` statement to map the exceptions data to the exceptions table. An exceptions `MAP` statement contains:
- (Required) The `INSERTALLRECORDS` option. This parameter converts all mapped operations to `INSERTS` so that all column values are mapped to the exceptions table.
 - (Required) The `EXCEPTIONSONLY` option. This parameter causes Replicat to map operations that generate an error, but not those that were successful.
 - (Optional) A `COLMAP` clause. If the names and definitions of the columns in the exceptions table are identical to those of the source table, and the exceptions table only contains those columns, no `COLMAP` is needed. However, if any names or definitions differ, or if there are extra columns in the exceptions table that you want to populate with additional data, use a `COLMAP` clause to map all columns.
- [Tools for Mapping Extra Data to the Exceptions Table](#)
 - [Sample Exceptions Mapping with Source and Target Columns Only](#)
 - [Sample Exceptions Mapping with Additional Columns in the Exceptions Table](#)

Tools for Mapping Extra Data to the Exceptions Table

The following are some tools that you can use in the `COLMAP` clause to populate extra columns:

- If the names and definitions of the source columns are identical to those of the target columns in the exceptions table, you can use the `USEDEFAULTS` keyword instead of explicitly mapping names. Otherwise, you must map those columns in the `COLMAP` clause, for example:

```
COLMAP (exceptions_col1 = col1, [...])
```

- To map the before image of the source row to columns in the exceptions table, use the @BEFORE conversion function, which captures the before image of a column from the trail record. This example shows the @BEFORE usage.

```
COLMAP (USEDEFAULTS, exceptions_coll1 = @BEFORE (source_coll1), &
exceptions_coll2 = @BEFORE (source_coll2), [...])
```

- To map the current image of the target row to columns in the exceptions table, use a SQLEXEC query to capture the image, and then map the results of the query to the columns in the exceptions table by using the 'queryID.column' syntax in the COLMAP clause, as in the following example:

```
COLMAP (USEDEFAULTS, name_current = queryID.name, phone_current =
queryID.phone, [...])
```

- To map timestamps, database errors, and other environmental information, use the appropriate Oracle GoldenGate column-conversion functions. For example, the following maps the current timestamp at time of execution.

```
res_date = @DATENOW ()
```

See [Sample Exceptions Mapping with Additional Columns in the Exceptions Table](#), for how to combine these features in a COLMAP clause in the exceptions MAP statement to populate a detailed exceptions table.

See Reference for Oracle GoldenGate for Windows and UNIX for the usage and syntax of the parameters and column-conversion functions shown in these examples.

Sample Exceptions Mapping with Source and Target Columns Only

The following is a sample parameter file that shows error handling and simple exceptions mapping for the source and target tables that are used in the CDR examples that begin. This example maps source and target columns, but no extra columns. For the following reasons, a COLMAP clause is not needed in the exceptions MAP statement in this example:

- The source and target exceptions columns are identical in name and definition.
- There are no other columns in the exceptions table.

Note:

This example intentionally leaves out other parameters that are required in a Replicat parameter file, such as process name and login credentials, as well as any optional parameters that may be required for a given database type. When using line breaks to split a parameter statement into multiple lines, use an ampersand (&) at the end of each line.

```
-- REPEROR error handling: DEFAULT represents all error types. DISCARD
-- writes operations that could not be processed to a discard file.
REPEROR (DEFAULT, DISCARD)
-- Specifies a discard file.
DISCARDFILE /users/ogg/discards/discards.dsc, PURGE
-- The regular MAP statement with the CDR parameters
MAP fin.src, TARGET fin.tgt, &
COMPARECOLS (ON UPDATE ALL, ON DELETE ALL), &
RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
```

```

RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)), &
);
-- Starts the exceptions MAP statement by mapping the source table to the
-- exceptions table.
MAP fin.src, TARGET fin.exception, &
-- directs Replicat only to map operations that caused the error specified
-- in REPEROR.
EXCEPTIONSONLY, &
-- directs Replicat to convert all the exceptions to inserts into the
-- exceptions table. This is why there cannot be a primary key constraint
-- on the exceptions table.
INSERTALLRECORDS
;

```

Sample Exceptions Mapping with Additional Columns in the Exceptions Table

The following is a sample parameter file that shows error handling and complex exceptions mapping for the source and target tables that are used in the CDR examples that begin. In this example, the exceptions table has the same rows as the source table, but it also has additional columns to capture context data.

Note:

This example intentionally leaves out other parameters that are required in a Replicat parameter file, such as process name and login credentials, as well as any optional parameters that may be required for a given database type. When using line breaks to split a parameter statement into multiple lines, use an ampersand (&) at the end of each line.

```

-- REPEROR error handling: DEFAULT represents all error types. DISCARD
-- writes operations that could not be processed to a discard file.
REPEROR (DEFAULT, DISCARD)
-- Specifies the discard file.
DISCARDFILE /users/ogg/discards/discards.dsc, PURGE
-- The regular MAP statement with the CDR parameters
MAP fin.src, TARGET fin.tgt, &
COMPARECOLS (ON UPDATE ALL, ON DELETE ALL), &
RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD))
);
-- Starts the exceptions MAP statement by mapping the source table to the --
exceptions table.
MAP fin.src, TARGET fin.exception, &
-- directs Replicat only to map operations that caused the error specified
-- in REPEROR.
EXCEPTIONSONLY, &
-- directs Replicat to convert all the exceptions to inserts into the
-- exceptions table. This is why there cannot be a primary key constraint
-- on the exceptions table.
INSERTALLRECORDS &

```

```

-- SQLEXEC query to select the values from the target record before the
-- Replicat statement is applied. These are mapped to the *_target
-- columns later.
SQLEXEC (id qry, query 'select name, phone, address, salary, balance, & comment,
last_mod_time from fin.tgt where name = :pl', PARAMS(pl = name)), &
-- Start of the column mapping, specifies use default column definitions.
COLMAP ( &
-- USEDEFAULTS maps the source columns to the target exceptions columns
-- that receive the after image that Replicat applied or tried to apply.
-- In this case, USEDEFAULTS can be used because the names and definitions
-- of the source and target exceptions columns are identical; otherwise
-- the columns must be mapped explicitly in the COLMAP clause.
USEDEFAULTS, &
-- captures the timestamp when the resolution was performed.
res_date = @DATENOW (), &
-- captures and maps the DML operation type.
optype = @GETENV ('LASTERR', 'OPTYPE'), &
-- captures and maps the database error number that was returned.
dberrnum = @GETENV ('LASTERR', 'DBERRNUM'), &
-- captures and maps the database error that was returned.
dberrmsg = @GETENV ('LASTERR', 'DBERRMSG'), &
-- captures and maps the name of the target table
tablename = @GETENV ('GGHEADER', 'TABLENAME'), &
-- If the names and definitions of the source columns and the target
-- exceptions columns were not identical, the columns would need to
-- be mapped in the COLMAP clause instead of using USEDEFAULTS, as
-- follows:
-- name_after = name, &
-- phone_after = phone, &
-- address_after = address, &
-- salary_after = salary, &
-- balance_after = balance, &
-- comment_after = comment, &
-- last_mod_time_after = last_mod_time &
-- maps the before image of each column from the trail to a column in the
-- exceptions table.
name_before = @BEFORE (name), &
phone_before = @BEFORE (phone), &
address_before = @BEFORE (address), &
salary_before = @BEFORE (salary), &
balance_before = @BEFORE (balance), &
comment_before = @BEFORE (comment), &
last_mod_time_before = @BEFORE (last_mod_time), &
-- maps the results of the SQLEXEC query to rows in the exceptions table
-- to show the current image of the row in the target.
name_current = qry.name, &
phone_current = qry.phone, &
address_current = qry.address, &
salary_current = qry.salary, &
balance_current = qry.balance, &
comment_current = qry.comment, &
last_mod_time_current = qry.last_mod_time)
;

```

For more information about creating an exceptions table and using exceptions mapping, see [Handling Replicat Errors during DML Operations](#).

Once you are confident that your routines work as expected in all situations, you can reduce the amount of data that is logged to the exceptions table to reduce the overhead of the resolution routines.

Viewing CDR Statistics

The CDR feature provides the following methods for viewing the results of conflict resolution.

- [Report File](#)
- [GGSCI](#)
- [Column-conversion Functions](#)

Report File

Replicat writes CDR statistics to the report file:

```
Total CDR conflicts          7
  CDR resolutions succeeded    6
  CDR resolutions failed      1
  CDR INSERTROWEXISTS conflicts 1
  CDR UPDATEROWEXISTS conflicts 4
  CDR UPDATEROWMISSING conflicts
  CDR DELETEROWEXISTS conflicts 1
  CDR DELETEROWMISSING conflicts 1
```

GGSCI

You can view CDR statistics from GGSCI by using the `STATS REPLICAT` command with the `REPORTCDR` option:

```
STATS REPLICAT group, REPORTCDR
```

Column-conversion Functions

The following CDR statistics can be retrieved and mapped to an exceptions table or used in other Oracle GoldenGate parameters that accept input from column-conversion functions, as appropriate.

- Number of conflicts that Replicat detected
- Number of resolutions that Replicat resolved
- Number of resolutions that Replicat could not resolve

To retrieve these statistics, use the `@GETENV` column-conversion function with the 'STATS' or 'DELTASTATS' information type. The results are based on the current Replicat session. If Replicat stops and restarts, it resets the statistics.

You can return these statistics for a specific table or set of wildcarded tables:

```
@GETENV ('STATS', 'TABLE', 'SCHEMA.TABLNAME', 'CDR_CONFLICTS')
@GETENV ('STATS', 'TABLE', 'SCHEMA.TABLNAME', 'CDR_RESOLUTIONS_SUCCEEDED')
@GETENV ('STATS', 'TABLE', 'SCHEMA.TABLNAME', 'CDR_RESOLUTIONS_FAILED')
```

You can return these statistics for all of the tables in all of the `MAP` statements in the Replicat parameter file:

```
@GETENV ('STATS', 'CDR_CONFLICTS')
@GETENV ('STATS', 'CDR_RESOLUTIONS_SUCCEEDED')
@GETENV ('STATS', 'CDR_RESOLUTIONS_FAILED')
```

The 'STATS' information type in the preceding examples can be replaced by 'DELTASTATS' to return the requested counts since the last execution of 'DELTASTATS'.

For more information about @GETENV, see *Reference for Oracle GoldenGate*.

CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD

This example resolves all conflict types by using the USEMAX, OVERWRITE, and DISCARD resolutions.

- [Table Used in this Example](#)
- [MAP Statement with Conflict Resolution Specifications](#)
- [Description of MAP Statement](#)
- [Error Handling](#)
- [INSERTROWEXISTS with the USEMAX Resolution](#)
- [UPDATEROWEXISTS with the USEMAX Resolution](#)
- [UPDATEROWMISSING with OVERWRITE Resolution](#)
- [DELETEROWMISSING with DISCARD Resolution](#)
- [DELETEROWEXISTS with OVERWRITE Resolution](#)

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
  name varchar2(30) primary key,
  phone varchar2(10),
  address varchar2(100),
  salary number,
  balance number,
  comment varchar2(100),
  last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement with Conflict Resolution Specifications

```
MAP fin.src, TARGET fin.tgt,
  COMPARECOLS (ON UPDATE ALL, ON DELETE ALL),
  RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
  RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
  RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)),
  RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)),
  RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)),
  );
```


Description of MAP Statement

The following describes the MAP statement:

- Per COMPARECOLS, use the before image of all columns in the trail record in the Replicat WHERE clause for updates and deletes.
- Per DEFAULT, use all columns as the column group for all conflict types; thus the resolution applies to all columns.
- For an INSERTROWEXISTS conflict, use the USEMAX resolution: If the row exists during an insert, use the last_mod_time column as the resolution column for deciding which is the greater value: the value in the trail or the one in the database. If the value in the trail is greater, apply the record but change the insert to an update. If the database value is higher, ignore the record.
- For an UPDATEROWEXISTS conflict, use the USEMAX resolution: If the row exists during an update, use the last_mod_time column as the resolution column: If the value in the trail is greater, apply the update.
- If you use USEMIN or USEMAX, and the values are exactly the same, then RESOLVECONFLICT isn't triggered and the incoming row is ignored. If you use USEMINEQ or USEMAXEQ, and the values are exactly the same, then the resolution is triggered.
- For a DELETEROWEXISTS conflict, use the OVERWRITE resolution: If the row exists during a delete operation, apply the delete.
- For an UPDATEROWMISSING conflict, use the OVERWRITE resolution: If the row does not exist during an update, change the update to an insert and apply it.
- For a DELETROWMISSING conflict use the DISCARD resolution: If the row does not exist during a delete operation, discard the trail record.

Note:

As an alternative to USEMAX, you can use the USEMAXEQ resolution to apply a >= condition. For more information, see *Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#).

INSERTROWEXISTS with the USEMAX Resolution

For this example, the USEMAX resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve an insert where the row exists in the source and target, but some or all row values are different.

Table 10-1 INSERTROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments
Before image in trail	None (row was inserted on the source).	N/A
After image in trail	name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 3:00'	last_mod_time='9/1/10 3:00' is the after image of the resolution column. Since there is an after image, this will be used to determine the resolution.
Target database image	name='Mary' phone='111111' address='Ralston' salary=200 balance=500 comment='aaa' last_mod_time='9/1/10 1:00'	last_mod_time='9/1/10 1:00' is the current image of the resolution column in the target against which the resolution column value in the trail is compared.
Initial INSERT applied by Replicat that detects the conflict	SQL bind variables: 1)'Mary' 2)'1234567890' 3)'Oracle Pkwy' 4)100 5)100 6)NULL 7)'9/1/10 3:00'	This SQL returns a uniqueness conflict on 'Mary'.
UPDATE applied by Replicat to resolve the conflict	SQL bind variables: 1)'1234567890' 2)'Oracle Pkwy' 3)100 4)100 5)NULL 6)'9/1/10 3:00' 7)'Mary' 8)'9/1/10 3:00'	Because USEMAX is specified for INSERTROWEXISTS, Replicat converts the insert to an update, and it compares the value of last_mod_time in the trail record with the value in the database. The value in the record is greater, so the after images for columns in the trail file are applied to the target.

UPDATEROWEXISTS with the USEMAX Resolution

For this example, the USEMAX resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve an update where the row exists in the source and target, but some or all row values are different.

Table 10-2 UPDATEROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments
Before image in trail	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 3:00'</pre>	last_mod_time='9/1/10 3:00' is the before image of the resolution column.
After image in trail	<pre>phone='222222' address='Holly' last_mod_time='9/1/10 5:00'</pre>	last_mod_time='9/1/10 5:00' is the after image of the resolution column. Since there is an after image, this will be used to determine the resolution.
Target database image	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=600 comment='com' last_mod_time='9/1/10 6:00'</pre>	last_mod_time='9/1/10 6:00' is the current image of the resolution column in the target against which the resolution column value in the trail is compared.
Initial UPDATE applied by Replicat that detects the conflict	<p>SQL bind variables:</p> <pre>1)'222222' 2)'Holly' 3)'9/1/10 5:00' 4)'Mary' 5)'1234567890' 6)'Oracle Pkwy' 7)100 8)100 9)NULL 10)'9/1/10 3:00'</pre>	<p>This SQL returns a no-data-found error because the values for the balance, comment, and last_mod_time are different in the target.</p> <p>All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL.</p>
UPDATE applied by Replicat to resolve the conflict	<p>SQL bind variables:</p> <pre>1)'Mary' 2)'222222' 3)'Holly' 4)100 5)100 6)NULL 7)'9/1/10 5:00' 8)'Mary' 9)'9/1/10 5:00'</pre>	<p>Because the after value of last_mod_time in the trail record is less than the current value in the database, the database value is retained. Replicat applies the operation with a WHERE clause that contains the primary key plus a last_mod_time value set to less than 9/1/10 5:00. No rows match this criteria, so the statement fails with a "data not found" error, but Replicat ignores the error because a USEMAX resolution is expected to fail if the condition is not satisfied.</p>

UPDATEROWMISSING with OVERWRITE Resolution

For this example, the OVERWRITE resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the

case where the target row is missing. The logical resolution, and the one used, is to overwrite the row into the target so that both databases are in sync again.

Table 10-3 UPDATEROWMISSING Conflict with OVERWRITE Resolution

Image	SQL	Comments
Before image in trail	<pre>name='Jane' phone='333' address='Oracle Pkwy' salary=200 balance=200 comment=NULL last_mod_time='9/1/10 7:00'</pre>	N/A
After image in trail	<pre>phone='4444' address='Holly' last_mod_time='9/1/10 8:00'</pre>	
Target database image	None (row for Jane is missing)	
Initial UPDATE applied by Replicat that detects the conflict	<p>SQL bind variables:</p> <pre>1)'4444' 2)'Holly' 3)'9/1/10 8:00' 4)'Jane' 5)'333' 6)'Oracle Pkwy' 7)200 8)200 9)NULL 10)'9/1/10 7:00'</pre>	This SQL returns a no-data-found error. All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL.
INSERT applied by Replicat to resolve the conflict	<p>SQL bind variables:</p> <pre>1)'Jane' 2)'4444' 3)'Holly' 4)200 5)200 6)NULL 7)'9/1/10 8:00'</pre>	The update is converted to an insert because OVERWRITE is the resolution. The after image of a column is used if available; otherwise the before image is used.

DELETEROWMISSING with DISCARD Resolution

For this example, the DISCARD resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the case where the target row is missing. In the case of a delete on the source, it is acceptable for the target row not to exist (it would need to be deleted anyway), so the resolution is to discard the DELETE operation that is in the trail.

Table 10-4 DELETEROWMSING Conflict with DISCARD Resolution

Image	SQL	Comments
Before image in trail	<pre>name='Jane' phone='4444' address='Holly' salary=200 balance=200 comment=NULL last_mod_time='9/1/10 8:00'</pre>	N/A
After image in trail	None	N/A
Target database image	None (row missing)	N/A
Initial DELETE applied by Replicat that detects the conflict	SQL bind variables: 1)'Jane' 2)'4444' 3)'Holly' 4)200 5)200 6)NULL 7)'9/1/10 8:00'	This SQL returns a no-data-found error. All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL.
SQL applied by Replicat to resolve the conflict	None	Because DISCARD is specified as the resolution for DELETEROWMISSING, so the delete from the trail goes to the discard file.

DELETEROWEXISTS with OVERWRITE Resolution

For this example, the OVERWRITE resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the case where the source row was deleted but the target row exists. In this case, the OVERWRITE resolution applies the delete to the target.

Table 10-5 DELETEROWEXISTS Conflict with OVERWRITE Resolution

Image	SQL	Comments
Before image in trail	<pre>name='Mary' phone='222222' address='Holly' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 5:00'</pre>	N/A
After image in trail	None	N/A

Table 10-5 (Cont.) DELETEROWEXISTS Conflict with OVERWRITE Resolution

Image	SQL	Comments
Target database image	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=600 comment=com last_mod_time='9/1/10 7:00'</pre>	The row exists on the target, but the phone, address, balance, comment, and last_mod_time columns are different from the before image in the trail.
Initial DELETE applied by Replicat that detects the conflict	<pre>SQL bind variables: 1)'Mary' 2)'222222' 3)'Holly' 4)100 5)100d 6)NULL 7)'9/1/10 5:00'</pre>	<p>All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL.</p> <p>A no-data-found error occurs because of the difference between the before and current values.</p>
DELETE applied by Replicat to resolve the conflict	<pre>SQL bind variables: 1)'Mary'</pre>	Because OVERWRITE is the resolution, the DELETE is applied using only the primary key (to avoid an integrity error).

CDR Example 2: UPDATEROWEXISTS with USEDELTA and USEMAX

This example resolves the condition where a target row exists on UPDATE but non-key columns are different, and it uses two different resolution types to handle this condition based on the affected column.

- [Table Used in this Example](#)
- [MAP Statement](#)
- [Description of MAP Statement](#)
- [Error Handling](#)

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
  name varchar2(30) primary key,
  phone varchar2(10),
  address varchar2(100),
  salary number,
  balance number,
  comment varchar2(100),
  last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement

```
MAP fin.src, TARGET fin.tgt,
  COMPARECOLS
  (ON UPDATE KEYINCLUDING (address, phone, salary, last_mod_time),
  ON DELETE KEYINCLUDING (address, phone, salary, last_mod_time)),
  RESOLVECONFLICT (
  UPDATEROWEXISTS,
  (delta_res_method, USEDELTA, COLS (salary)),
  (DEFAULT, USEMAX (last_mod_time)));
```

Description of MAP Statement

For an UPDATEROWEXISTS conflict, where a target row exists on UPDATE but non-key columns are different, use two different resolutions depending on the column:

- Per the `delta_res_method` resolution, use the USEDELTA resolution logic for the `salary` column so that the change in value will be added to the current value of the column.
- Per DEFAULT, use the USEMAX resolution logic for all other columns in the table (the default column group), using the `last_mod_time` column as the resolution column. This column is updated with the current time whenever the row is modified; the value of this column in the trail is compared to the value in the target. If the value of `last_mod_time` in the trail record is greater than the current value of `last_mod_time` in the target database, the changes to `name`, `phone`, `address`, `balance`, `comment` and `last_mod_time` are applied to the target.

Per COMPARECOLS, use the primary key (`name` column) plus the `address`, `phone`, `salary`, and `last_mod_time` columns as the comparison columns for conflict detection for UPDATE and DELETE operations. (The `balance` and `comment` columns are not compared.)

Note:

As an alternative to USEMAX, you can use the USEMAXEQ resolution to apply a >= condition. For more information, see *Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#).

Table 10-6 UPDATEROWEXISTS with USEDELTA and USEMAX

Image	SQL	Comments
Before image in trail	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 3:00'</pre>	<p>last_mod_time='9/1/10 3:00' is the before image of the resolution column for the USEMAX resolution.</p> <p>salary=100 is the before image for the USEDELTA resolution.</p>
After image in trail	<pre>phone='222222' address='Holly' salary=200 comment='new' last_mod_time='9/1/10 5:00'</pre>	<p>last_mod_time='9/1/10 5:00' is the after image of the resolution column for USEMAX. Since there is an after image, this will be used to determine the resolution.</p>
Target database image	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=600 balance=600 comment='com' last_mod_time='9/1/10 4:00'</pre>	<p>last_mod_time='9/1/10 4:00' is the current image of the resolution column in the target against which the resolution column value in the trail is compared.</p> <p>salary=600 is the current image of the target column for the USEDELTA resolution.</p>
Initial UPDATE applied by Replicat that detects the conflict	<p>SQL bind variables:</p> <pre>1)'222222' 2)'Holly' 3)200 4)'new' 5)'9/1/10 5:00' 6)'Mary' 7)'1234567890' 8)'Oracle Pkwy' 9)100 10)'9/1/10 3:00'</pre>	<p>This SQL returns a no-data-found error because the values for the salary and last_mod_time are different. (The values for comment and balance are also different, but these columns are not compared.)</p>
UPDATE applied by Replicat to resolve the conflict for salary, using USEDELTA.	<p>SQL bind variables:</p> <pre>1)200 2)100 3)'Mary'</pre>	<p>Per USEDELTA, the difference between the after image of salary (200) in the trail and the before image of salary (100) in the trail is added to the current value of salary in the target (600). The result is 700.</p> <p style="text-align: center;">$600 + (200 - 100) = 700$</p>
UPDATE applied by Replicat to resolve the conflict for the default columns, using USEMAX.	<p>SQL bind variables:</p> <pre>1)'222222' 2)'Holly' 3)'new' 4)'9/1/10 5:00' 5)'Mary' 6)'9/1/10 5:00'</pre>	<p>Per USEMAX, because the after value of last_mod_time in the trail record is greater than the current value in the database, the row is updated with the after values from the trail record.</p> <p>Note that the salary column is not set here, because it is resolved with the UPDATE from the USEDELTA resolution.</p>

CDR Example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

This example resolves the conflict where a target row exists on UPDATE but non-key columns are different, and it uses three different resolution types to handle this condition based on the affected column.

- [Table Used in this Example](#)
- [MAP Statement](#)
- [Description of MAP Statement](#)
- [Error Handling](#)

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
  name varchar2(30) primary key,
  phone varchar2(10),
  address varchar2(100),
  salary number,
  balance number,
  comment varchar2(100),
  last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement

```
MAP fin.src, TARGET fin.tgt,
COMPARECOLS
  (ON UPDATE ALLEXCLUDING (comment)),
RESOLVECONFLICT (
  UPDATEROWEXISTS,
  (delta_res_method, USEDELTA, COLS (salary, balance)),
  (max_res_method, USEMAX (last_mod_time), COLS (address, last_mod_time)),
  (DEFAULT, IGNORE));
```

Description of MAP Statement

- For an UPDATEROWEXISTS conflict, where a target row exists on UPDATE but non-key columns are different, use two different resolutions depending on the column:
 - Per the `delta_res_method` resolution, use the USEDELTA resolution logic for the `salary` and `balance` columns so that the change in each value will be added to the current value of each column.
 - Per the `max_res_method` resolution, use the USEMAX resolution logic for the `address` and `last_mod_time` columns. The `last_mod_time` column is the resolution column. This column is updated with the current time whenever the

row is modified; the value of this column in the trail is compared to the value in the target. If the value of `last_mod_time` in the trail record is greater than the current value of `last_mod_time` in the target database, the changes to `address` and `last_mod_time` are applied to the target; otherwise, they are ignored in favor of the target values.

- Per `DEFAULT`, use the `IGNORE` resolution logic for the remaining columns (`phone` and `comment`) in the table (the default column group). Changes to these columns will always be ignored by Replicat.
- Per `COMPARECOLS`, use all columns except the `comment` column as the comparison columns for conflict detection for `UPDATE` operations. `comment` will not be used in the `WHERE` clause for updates, but all other columns that have a before image in the trail record will be used.

Note:

As an alternative to `USEMAX`, you can use the `USEMAXEQ` resolution to apply a `>=` condition. For more information, see *Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see [Configuring the Oracle GoldenGate Parameter Files for Error Handling](#).

Table 10-7 UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments
Before image in trail	<pre>name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 3:00'</pre>	<p><code>last_mod_time='9/1/10 3:00</code> is the before image of the resolution column for the <code>USEMAX</code> resolution.</p> <p><code>salary=100</code> and <code>balance=100</code> are the before images for the <code>USEDELTA</code> resolution.</p>
After image in trail	<pre>phone='222222' address='Holly' salary=200 comment='new' last_mod_time='9/1/10 5:00'</pre>	<p><code>last_mod_time='9/1/10 5:00</code> is the after image of the resolution column for <code>USEMAX</code>. Since there is an after image, this will be used to determine the resolution.</p> <p><code>salary=200</code> is the only after image available for the <code>USEDELTA</code> resolution. For <code>balance</code>, the before image will be used in the calculation.</p>

Table 10-7 (Cont.) UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments
Target database image	<pre>name='Mary' phone='1234567890' address='Ralston' salary=600 balance=600 comment='com' last_mod_time='9/1/10 4:00'</pre>	<p>last_mod_time='9/1/10 4:00' is the current image of the resolution column in the target against which the resolution column value in the trail is compared for USEMAX.</p> <p>salary=600 and balance=600 are the current images of the target columns for USEDELTA.</p>
Initial UPDATE applied by Replicat that detects the conflict	<p>SQL bind variables:</p> <pre>1)'222222' 2)'Holly' 3)200 4)'new' 5)'9/1/10 5:00' 6)'Mary' 7)'1234567890' 8)'Oracle Pkwy' 9)100 10)100 11)'9/1/10 3:00'</pre>	<p>This SQL returns a no-data-found error because the values for the address, salary, balance and last_mod_time columns are different.</p>
UPDATE applied by Replicat to resolve the conflict for salary, using USEDELTA.	<p>SQL bind variables:</p> <pre>1)200 2)100 3)'Mary'</pre>	<p>For salary, there is a difference of 100, but there was no change in value for balance, so it is not needed in the update SQL. Per USEDELTA, the difference (delta) between the after (200) image and the before image (100) of salary in the trail is added to the current value of salary in the target (600). The result is 700.</p>
UPDATE applied by Replicat to resolve the conflict for USEMAX.	<p>SQL bind variables:</p> <pre>1)'Holly' 2)'9/1/10 5:00' 3)'Mary' 4)'9/1/10 5:00'</pre>	<p>Because the after value of last_mod_time in the trail record is greater than the current value in the database, that column plus the address column are updated with the after values from the trail record.</p> <p>Note that the salary column is not set here, because it is resolved with the UPDATE from the USEDELTA resolution.</p>
UPDATE applied by Replicat for IGNORE.	<p>SQL bind variables:</p> <pre>1)'222222' 2)'new' 3)'Mary'</pre>	<p>IGNORE is specified for the DEFAULT column group (phone and comment), so no resolution SQL is applied.</p>

11

Mapping and Manipulating Data

This chapter describe how you can integrate data between source and target tables.

Topics:

- [Limitations of Support](#)
- [Parameters that Control Mapping and Data Integration](#)
- [Mapping between Dissimilar Databases](#)
- [Deciding Where Data Mapping and Conversion Will Take Place](#)
- [Globalization Considerations when Mapping Data](#)
- [Mapping Columns](#)
- [Selecting and Filtering Rows](#)
- [Retrieving Before and After Values](#)
- [Selecting Columns](#)
- [Selecting and Converting SQL Operations](#)
- [Using Transaction History](#)
- [Testing and Transforming Data](#)
- [Using Tokens](#)

Limitations of Support

The following are limitations to the support of data mapping and manipulation.

- Oracle GoldenGate does not support the filtering, column mapping, or manipulation of large objects.
- Some Oracle GoldenGate features and functionality do not support the use of data filtering and manipulation. Where applicable, this limitation is documented.

Parameters that Control Mapping and Data Integration

All data selection, mapping, and manipulation that Oracle GoldenGate performs is accomplished by using one or more options of the `TABLE` and `MAP` parameters.

- Use `TABLE` in the Extract parameter file.
- Use `MAP` in the Replicat parameter file.

`TABLE` and `MAP` specify the database objects that are affected by the other parameters in the parameter file. See [Specifying Object Names in Oracle GoldenGate Input](#) for instructions for specifying object names in these parameters.

Mapping between Dissimilar Databases

Mapping and conversion between tables that have different data structures requires either a source-definitions file, a target-definitions file, or in some cases both. When used, this file must be specified with the `SOURCEDEFS` or `TARGETDEFS` parameter.

This is not applicable if you are using self-describing trail files.

For more information about how to create a source-definitions or target-definitions file, see [Associating Replicated Data with Metadata](#).

Deciding Where Data Mapping and Conversion Will Take Place

If the configuration you are planning involves a large amount of column mapping or data conversion, observe the following guidelines to determine which process or processes will perform these functions.

- [Mapping and Conversion on Windows and UNIX Systems](#)
- [Mapping and Conversion on NonStop Systems](#)

Mapping and Conversion on Windows and UNIX Systems

When Oracle GoldenGate is operating only on Windows-based and UNIX-based systems, column mapping and conversion can be performed on the source system, on the target system, or on an intermediary system. To prevent the added overhead of this processing on the source system, you can configure the mapping and conversion to be performed on the target system or on an intermediary system.

In the case where there are multiple sources and one target, it might be more efficient to perform the mapping and conversion on the source. You can use one target-definitions file generated from the target tables, rather than having to manage an individual source-definitions file for each source database, which needs to be copied to the target each time the applications make layout changes.

For more information on which types of definitions files to use, and where, see [Associating Replicated Data with Metadata](#).

Mapping and Conversion on NonStop Systems

If you are mapping or converting data from a Windows or UNIX system to a NonStop Enscribe target, the mapping or conversion must be performed on the Windows or UNIX source system. Replicat for NonStop cannot convert three-part or two-part SQL table names and data types to the three-part file names that are used for the Enscribe platform. Extract can format the trail data with Enscribe names and target data types.

Globalization Considerations when Mapping Data

When planning to map and convert data between databases and platforms, take into consideration what is supported or not supported by Oracle GoldenGate in terms of globalization.

Topics:

- [Conversion between Character Sets](#)
- [Preservation of Locale](#)
- [Support for Escape Sequences](#)

Conversion between Character Sets

Oracle GoldenGate converts between source and target character sets if they are different, so that object names and column data are compared, mapped, and manipulated properly from one database to another. See [Supported Character Sets](#), for a list of supported character sets.

To ensure accurate character representation from one database to another, the following must be true:

- The character set of the target database must be a superset or equivalent of the character set of the source database. *Equivalent* means not equal, but having the same set of characters. For example, Shift-JIS and EUC-JP technically are not completely equal, but have the same characters in most cases.
- If your client applications use different character sets, the database character set must also be a superset or equivalent of the character sets of the client applications.

In this configuration, every character is represented when converting from a client or source character set to the local database character set.

A Replicat process can support conversion from one source character set to one target character set.

- [Database Object Names](#)
- [Column Data](#)

Database Object Names

Oracle GoldenGate processes catalog, schema, table and column names in their native language as determined by the character set encoding of the source and target databases. This support preserves single-byte and multibyte names, symbols, accent characters, and case-sensitivity with locale taken into account where available, at all levels of the database hierarchy.

Column Data

Oracle GoldenGate supports the conversion of column data between character sets when the data is contained in the following column types:

- Character-type columns: CHAR/VARCHAR/CLOB to CHAR/VARCHAR/CLOB of another character set; and CHAR/VARCHAR/CLOB to and from NCHAR/NVARCHAR/NCLOB.
- Columns that contain string-based numbers and date-time data. Conversions of these columns is performed between z/OS EBCDIC and non-z/OS ASCII data. Conversion is not performed between ASCII and ASCII versions of this data, nor between EBCDIC and EBCDIC versions, because the data are compatible in these cases.

 **Note:**

Oracle GoldenGate supports timestamp data from 0001-01-03 00:00:00 to 9999-12-31 23:59:59. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the timezone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.

Character-set conversion for column data is limited to a direct mapping of a source column and a target column in the `COLMAP` or `USEDEFAULTS` clauses of the `Replicat MAP` parameter. A direct mapping is a name-to-name mapping without the use of a stored procedure or column-conversion function. `Replicat` performs the character-set conversion. No conversion is performed by `Extract` or a data pump.

If the trail is written by a version of `Extract` that is prior to version 11.2.1, the character set for character-type columns must be supplied to `Replicat` with the `SOURCECHARSET` parameter. For more information, see *Reference for Oracle GoldenGate*.

Preservation of Locale

Oracle GoldenGate takes the locale of the database into account when comparing case-insensitive object names. See [Supported Locales](#) for a list of supported locales.

Support for Escape Sequences

Oracle GoldenGate supports the use of an escape sequence to represent a string column, literal text, or object name in the parameter file. You can use an escape sequence if the operating system does not support the required character, such as a control character, or for any other purpose that requires a character that cannot be used in a parameter file.

An escape sequence can be used anywhere in the parameter file, but is particularly useful in the following elements within a `TABLE` or `MAP` statement:

- An object name
- `WHERE` clause
- `COLMAP` clause to assign a Unicode character to a Unicode column, or to assign a native-encoded character to a column.
- Oracle GoldenGate column conversion functions within a `COLMAP` clause.

Oracle GoldenGate supports the following types of escape sequence:

- `\uFFFF` Unicode escape sequence. Any `UNICODE` code point can be used except surrogate pairs.
- `\377` Octal escape sequence
- `\xFF` Hexadecimal escape sequence

The following rules apply:

- If used for mapping of an object name in `TABLE` or `MAP`, no restriction apply. For example, the following `TABLE` specification is valid:

```
TABLE schema."u3000ABC";
```

- If used with a column-mapping function, any code point can be used, but only for an NCHAR/NVARCHAR column. For an CHAR/VARCHAR column, the code point is limited to the equivalent of 7-bit ASCII.
- The source and target data types must be identical (for example, NCHAR to NCHAR).
- Begin each escape sequence with a reverse solidus (code point U+005C), followed by the character code point. (A solidus is more commonly known as the backslash symbol.) Use the escape sequence, instead of the actual character, within your input string in the parameter statement or column-conversion function.


 **Note:**

To specify an actual backslash in the parameter file, specify a double backslash. For example, the following finds a backslash in COL1: @STRFIND (COL1, '\\\ ').

To Use the \uFFFF Unicode Escape Sequence

- The \uFFFF Unicode escape sequence must begin with a lowercase u, followed by exactly four hexadecimal digits.
- Supported ranges are as follows:
 - 0 to 9 (U+0030 to U+0039)
 - A to F (U+0041 to U+0046)
 - a to f (U+0061 to U+0066)

\u20ac is the Unicode escape sequence for the Euro currency sign.

 **Note:**

For reliable cross-platform support, use the Unicode escape sequence. Octal and hexadecimal escape sequences are not standardized on different operating systems.

To Use the \377 Octal Escape Sequence

- Must contain exactly three octal digits.
 - Supported ranges:
 - Range for first digit is 0 to 3 (U+0030 to U+0033)
 - Range for second and third digits is 0 to 7 (U+0030 to U+0037)
- \200 is the octal escape sequence for the Euro currency sign on Microsoft Windows

To Use the \xFF Hexadecimal Escape Sequence

- Must begin with a lowercase x followed by exactly two hexadecimal digits.
- Supported ranges:

- 0 to 9 (U+0030 to U+0039)
- A to F (U+0041 to U+0046)
- a to f (U+0061 to U+0066)

\x80 is the hexadecimal escape sequence for the Euro currency sign on Microsoft Windows 1252 Latin1 code page.

Mapping Columns

Oracle GoldenGate provides for column mapping at the table level and at the global level. Default column mapping is also provided in the absence of explicit column mapping rules.

This section contains the following guidelines for mapping columns:

Topics:

- [Supporting Case and Special Characters in Column Names](#)
- [Configuring Table-level Column Mapping with COLMAP](#)
- [Configuring Global Column Mapping with COLMATCH](#)
- [Understanding Default Column Mapping](#)
- [Mapping Data Types from Column to Column](#)

Supporting Case and Special Characters in Column Names

By default, Oracle GoldenGate follows SQL-92 rules for specifying column names and literals. In Oracle GoldenGate parameter files, conversion functions, user exits, and commands, case-sensitive column names must be enclosed within double quotes if double quotes are required by the database to enforce case-sensitivity. For other case-sensitive databases that do not require quotes, case-sensitive column names must be specified as they are stored in the database. Literals must be enclosed within single quotes. See [Differentiating Case-Sensitive Column Names from Literals](#) for more information.

Configuring Table-level Column Mapping with COLMAP

Use the COLMAP option of the MAP and TABLE parameters to:

- map individual source columns to target columns that have different names.
- specify default column mapping when an explicit column mapping is not needed.
- Provide instructions for selecting, mapping, translating, and moving data from a source column into a target column.
- [Specifying the Columns to be Mapped in the COLMAP Clause](#)
- [Using USEDEFAULTS to Enable Default Column Mapping](#)
- [Determining Whether COLMAP Requires a Data-definitions File](#)

Specifying the Columns to be Mapped in the COLMAP Clause

The COLMAP syntax is the following:

```
COLMAP ([USEDEFAULTS, ] target_column = source_expression)
```

In this syntax, *target_column* is the name of the target column, and *source_expression* can be any of the following, allowing you to map the source column by name, so as to pass the source value exactly as recorded in the trail, or to transform the data before passing it to the target column:

- The name of a source column, such as `ORD_DATE`.
- Numeric constant, such as `123`.
- String constant enclosed within single quotes, such as `'ABCD'`.
- An expression using an Oracle GoldenGate column-conversion function. Within a `COLMAP` statement, you can employ any of the Oracle GoldenGate column-conversion functions to transform data for the mapped columns, for example:

```
@STREXT (COL1, 1, 3)
```

If the column mapping involves case-sensitive columns from different database types, specify each column as it is stored in the database.

- If the database requires double quotes to enforce case-sensitivity, specify the case-sensitive column name within double quotes.
- If the database is case-sensitive without requiring double quotes, specify the column name as it is stored in the database.

The following shows a mapping between a target column in an Oracle database and a source column in a case-sensitive SQL Server database.

```
COLMAP ("Cola" = Cola)
```

See [Specifying Object Names in Oracle GoldenGate Input](#) for more information about specifying names to Oracle GoldenGate.

See [Globalization Considerations when Mapping Data](#) for globalization considerations when mapping source and target columns in databases that have different character sets and locales.

Avoid using `COLMAP` to map a value to a key column (which causes the operation to become a primary key update). The `WHERE` clause that Oracle GoldenGate uses to locate the target row will not use the correct before image of the key column. Instead, it will use the after image. This will cause errors if you are using any functions based on that key column, such as a `SQLEXEC` statement, as shown in the following example.

- Source table `TCUSTOMER1`
- Target table `TCUSTOMER2`
- Column layout, both tables:
 - Column 1 = `Cust`
 - Column 2 = `Name`
 - Column 3 = `City`
 - Column 4 = `State`
- Primary key consists of the `Cust`, `Name`, and `City` columns.
- `SQLEXEC` query in the `MAP` statement:

```
SQLEXEC (id mytest, query 'select city from TCUSTOMER1 WHERE state = 'CA'',
noparams, ERROR RAISE)
```

- COLMAP statement in the MAP statement:

```
COLMAP ( usedefaults, city = mytest.city )
```

This is the sequence of events that causes the error:

1. INSERT statement inserts the following:

```
INSERT into TCUSTOMER1 values (Cust = '1234', Name = 'Ace', City = 'SF', State =
'CA');
Commit;
```

The SQLEXEC query returns the correct value, and the target table also has a value of SF for City and CA for State.

```
mytest.city = 'SF'
```

2. UPDATE statement changes City from SF to LA on the source. This does not succeed on the target. The SQLEXEC query looks up the City column in TCUSTOMER1 and returns a value of LA. Based on the COLMAP clause, the before and after versions of City both are now LA. This generates SQL error 1403 when executing the target WHERE clause, because a value of LA does not exist for the City column in the target table.

Using USEDEFAULTS to Enable Default Column Mapping

You can use the USEDEFAULTS option of COLMAP to specify automatic default column mapping for any corresponding source and target columns that have identical names. USEDEFAULTS can save you time by eliminating the need to map every target column explicitly.

Default mapping causes Oracle GoldenGate to map those columns and, if required, translate the data types based on the data-definitions file (see [Determining Whether COLMAP Requires a Data-definitions File](#)). Do not specify default mapping for columns that are mapped already with an explicit mapping statement.

The following example of a column mapping illustrates the use of both default and explicit column mapping for a source table ACCTBL and a target table ACCTTAB. Most columns are the same in both tables, except for the following differences:

- The source table has a CUST_NAME column, whereas the target table has a NAME column.
- A ten-digit PHONE_NO column in the source table corresponds to separate AREA_CODE, PHONE_PREFIX, and PHONE_NUMBER columns in the target table.
- Separate YY, MM, and DD columns in the source table correspond to a single TRANSACTION_DATE column in the target table.

To address those differences, USEDEFAULTS is used to map the similar columns automatically, while explicit mapping and conversion functions are used for dissimilar columns.

Table 11-1 Sample Column Mapping

Parameter statement	Description
MAP SALES.ACCTBL, TARGET SALES.ACCTTAB,	Maps the source table ACCTBL to the target table ACCTTAB.
COLMAP(Begins the COLMAP statement.
USEDEFAULTS,	Maps source columns as-is when the target column names are identical.
NAME = CUST_NAME,	Maps the source column CUST_NAME to the target column NAME.
TRANSACTION_DATE = @DATE ('YYYY-MM-DD', 'YY', YEAR, 'MM', MONTH, 'DD', DAY),	Converts the transaction date from the source date columns to the target column TRANSACTION_DATE by using the @DATE column conversion function.
AREA_CODE = @STREXT (PHONE_NO, 1, 3), PHONE_PREFIX = @STREXT (PHONE_NO, 4, 6), PHONE_NUMBER = @STREXT (PHONE_NO, 7, 10)) ;	Converts the source column PHONE_NO into the separate target columns of AREA_CODE, PHONE_PREFIX, and PHONE_NUMBER by using the @STREXT column conversion function.

See [Understanding Default Column Mapping](#) for more information about the rules followed by Oracle GoldenGate for default column mapping.

Determining Whether COLMAP Requires a Data-definitions File

When using COLMAP, you might need to create a data-definitions file. To make this determination, you must consider whether the source and target column structures are identical, as defined by Oracle GoldenGate.

For source and target structures to be identical, they must:

- be of the same database type, that is, all Oracle.
- have the same character set and locale.
- contain the same number of columns.
- have identical column names (including case, white spaces, and quotes if applicable).
- have identical data types.
- have identical column lengths.
- have the same column length semantics for character columns (bytes versus characters).
- define all of the columns in the same order.

When using COLMAP for source and target tables that are not identical in structure, you must:

- generate data definitions for the source tables, the target tables, or both, depending on the Oracle GoldenGate configuration and the databases that are being used.
- transfer the definitions file to the system where they will be used.
- use the `SOURCEDEFS` parameter to identify the definitions file for Replicat on a target system or use the `TARGETDEFS` parameter to identify the definitions file for Extract or a data pump on a source system or intermediary system.

When using `COLMAP` for source and target tables that are identical in structure, and you are only using `COLMAP` for other functions such as conversion, a source definitions file is not needed. When a definitions file is not being used, you must use the `ASSUMETARGETDEFS` parameter instead, unless you are using self-describing trail files. See *Reference for Oracle GoldenGate* for more information.

See [Associating Replicated Data with Metadata](#) for more information about using a definitions file.

Configuring Global Column Mapping with COLMATCH

Use the `COLMATCH` parameter to create global rules for column mapping. With `COLMATCH`, you can map between similarly structured tables that have different column names for the same sets of data. `COLMATCH` provides a more convenient way to map columns of this type than does using table-level mapping with a `COLMAP` clause in individual `TABLE` or `MAP` statements.

Case-sensitivity is supported as follows:

- For MySQL, SQL Server, and Teradata, if the database is case-sensitive, `COLMATCH` looks for an exact case and name match regardless of whether or not a name is specified in quotes.
- For Oracle Database and DB2 databases, where names can be either case-sensitive or case-insensitive in the same database and double quotes are required to show case-sensitivity, `COLMATCH` requires an exact case and name match when a name is in quotes in the database.

See [Specifying Object Names in Oracle GoldenGate Input](#) for more information about case-sensitivity support.

Syntax

```
COLMATCH
{NAMES target_column = source_column |
PREFIX prefix |
SUFFIX suffix |
RESET}
```

Table 11-2 COLMATCH Options

Argument	Description
<code>NAMES <i>target_column</i> = <i>source_column</i></code>	<p>Maps based on column names.</p> <p>Put double quotes around the column name if it is case-sensitive and the database requires quotes to enforce case-sensitivity. For these database types, an unquoted column name is treated as case-insensitive by Oracle GoldenGate.</p> <p>For databases that support case-sensitivity without requiring quotes, specify the column name as it is stored in the database.</p> <p>If the COLMATCH is between columns in different database types, make certain the names reflect the appropriate case representation for each one. For example, the following specifies a case-sensitive target column name "aBc" in an Oracle Database and a case-sensitive source column name aBc in a case-sensitive SQL Server database.</p> <pre>COLMATCH NAMES "aBc" = aBc</pre>
<code>PREFIX <i>prefix</i> SUFFIX <i>suffix</i></code>	<p>Ignores the specified name prefix or suffix.</p> <p>Put double quotes around the prefix or suffix if the database requires quotes to enforce case-sensitivity, for example "P_". For those database types, an unquoted prefix or suffix is treated as case-insensitive.</p> <p>For databases that support case-sensitivity without requiring quotes, specify the prefix or suffix as it is stored in the database. For example, P_ specifies a capital P prefix.</p> <p>The following example specifies a case-insensitive prefix to ignore. The target column name P_ABC is mapped to source column name ABC, and target column name P_abc is mapped to source column name abc.</p> <pre>COLMATCH PREFIX p_</pre> <p>The following example specifies a case-sensitive <i>suffix</i> to ignore. The target column name ABC_k is mapped to the source column name ABC, and the target column name "abc_k" is mapped to the source column name "abc".</p> <pre>SUFFIX "_k"</pre>
<code>RESET</code>	Turns off previously defined COLMATCH rules for subsequent TABLE or MAP statements.

The following example illustrates when to use COLMATCH. The source and target tables are identical except for slightly different table and column names. The database is case-insensitive.

Table 11-3 COLMATCH Example Table: Source Database

ACCT Table	ORD Table
CUST_CODE	CUST_CODE
CUST_NAME	CUST_NAME
CUST_ADDR	ORDER_ID
PHONE	ORDER_AMT
S_REP	S_REP
S_REPCODE	S_REPCODE

Table 11-4 COLMATCH Example Table: Target Database

ACCOUNT Table	ORDER Table
CUSTOMER_CODE	CUSTOMER_CODE
CUSTOMER_NAME	CUSTOMER_NAME
CUSTOMER_ADDRESS	ORDER_ID
PHONE	ORDER_AMT
REP	REP
REPCODE	REPCODE

To map the source columns to the target columns in this example, as well as to handle subsequent maps for other tables, the syntax is:

```
COLMATCH NAMES CUSTOMER_CODE = CUST_CODE
COLMATCH NAMES CUSTOMER_NAME = CUST_NAME
COLMATCH NAMES CUSTOMER_ADDRESS = CUST_ADDR
COLMATCH PREFIX S_
MAP SALES.ACCT, TARGET SALES.ACCOUNT, COLMAP (USEDEFAULTS);
MAP SALE.ORD, TARGET SALES.ORDER, COLMAP (USEDEFAULTS);
COLMATCH RESET
MAP SALES.REG, TARGET SALE.REG;
MAP SALES.PRICE, TARGET SALES.PRICE;
```

Based on the rules in the example, the following occurs:

- Data is mapped from the CUST_CODE columns in the source ACCT and ORD tables to the CUSTOMER_CODE columns in the target ACCOUNT and ORDER tables.
- The S_ prefix will be ignored.
- Columns with the same names, such as the PHONE and ORDER_AMT columns, are automatically mapped by means of USEDEFAULTS without requiring explicit rules. See [Understanding Default Column Mapping](#) for more information.
- The previous global column mapping is turned off for the tables REG and PRICE. Source and target columns in those tables are automatically mapped because all of the names are identical.

Understanding Default Column Mapping

If an explicit column mapping does not exist, either by using `COLMATCH` or `COLMAP`, Oracle GoldenGate maps source and target columns by default according to the following rules.

- If a source column is found whose name and case exactly match those of the target column, the two are mapped.
- If no case match is found, fallback name mapping is used. Fallback mapping performs a case-insensitive target table mapping to find a name match. Inexact column name matching is applied using upper cased names. This behavior is controlled by the `GLOBALS` parameter `NAMEMATCHIGNORECASE`. You can disable fallback name matching with the `NAMEMATCHEXACT` parameter, or you can keep it enabled but with a warning message by using the `NAMEMATCHNOWARNING` parameter.
- Target columns that do not correspond to any source column take default values determined by the database.

If the default mapping cannot be performed, the target column defaults to one of the values shown in [Table 11-5](#).

Table 11-5 Defaults Values for Target Columns

Column Type	Value
Numeric	Zero (0)
Character or VARCHAR	Spaces
Date or Datetime	Current date and time
Columns that can take a NULL value	Null

Mapping Data Types from Column to Column

The following explains how Oracle GoldenGate maps data types.

- [Numeric Columns](#)
- [Character-type Columns](#)
- [Datetime Columns](#)

Numeric Columns

Numeric columns are converted to match the type and scale of the target column. If the scale of the target column is smaller than that of the source, the number is truncated on the right. If the scale of the target column is larger than that of the source, the number is padded with zeros on the right.

You can specify a substitution value for invalid numeric data encountered when mapping number columns by using the `REPLACEBADNUM` parameter. See *Reference for Oracle GoldenGate* for more information.

Character-type Columns

Character-type columns can accept character-based data types such as `VARCHAR`, numeric in string form, date and time in string form, and string literals. If the scale of the target column is smaller than that of the source, the column is truncated on the right. If the scale of the target column is larger than that of the source, the column is padded with spaces on the right.

Literals must be enclosed within single quotes.

You can control the response of the Oracle GoldenGate process when a valid code point does not exist for either the source or target character set when mapping character columns by using the `REPLACEBADCHAR` parameter. See *Reference for Oracle GoldenGate* for more information.

Datetime Columns

Datetime (`DATE`, `TIME`, and `TIMESTAMP`) columns can accept datetime and character columns, as well as string literals. Literals must be enclosed within single quotes. To map a character column to a datetime column, make certain it conforms to the Oracle GoldenGate external SQL format of `YYYY-MM-DD HH:MI:SS.FFFFFFFF`.

Oracle GoldenGate supports timestamp data from `0001-01-03 00:00:00` to `9999-12-31 23:59:59`. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the timezone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.

Required precision varies according to the data type and target platform. If the scale of the target column is smaller than that of the source, data is truncated on the right. If the scale of the target column is larger than that of the source, the column is extended on the right with the values for the current date and time.

Selecting and Filtering Rows

To filter out or select rows for extraction or replication, use the `FILTER` and `WHERE` clauses of the `TABLE` and `MAP` parameters.

The `FILTER` clause offers you more functionality than the `WHERE` clause because you can employ any of the Oracle GoldenGate column conversion functions, whereas the `WHERE` clause accepts basic `WHERE` operators.

- [Selecting Rows with a FILTER Clause](#)
- [Selecting Rows with a WHERE Clause](#)
- [Considerations for Selecting Rows with FILTER and WHERE](#)

Selecting Rows with a FILTER Clause

Use a `FILTER` clause to select rows based on a numeric value by using basic operators or one or more Oracle GoldenGate column-conversion functions.

 **Note:**

To filter a column based on a string, use one of the Oracle GoldenGate string functions or use a `WHERE` clause.

The syntax for `FILTER` in a `TABLE` statement is as follows:

```
TABLE source_table,  
  , FILTER (  
  [, ON INSERT | ON UPDATE | ON DELETE]  
  [, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]  
  , filter_clause);
```

The syntax for `FILTER` in a `MAP` statement is as follows and includes an error-handling option.

```
MAP source_table, TARGET target_table,  
  , FILTER (  
  [, ON INSERT | ON UPDATE | ON DELETE]  
  [, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]  
  [, RAISEERROR error_number]  
  , filter_clause);
```

Valid `FILTER` clause elements are the following:

- An Oracle GoldenGate column-conversion function. These functions are built into Oracle GoldenGate so that you can perform tests, manipulate data, retrieve values, and so forth. See [Testing and Transforming Data](#) for more information about Oracle GoldenGate conversion functions.
- Numbers
- Columns that contain numbers
- Functions that return numbers
- Arithmetic operators:
 - + (plus)
 - - (minus)
 - * (multiply)
 - / (divide)
 - \ (remainder)
- Comparison operators:
 - > (greater than)
 - >= (greater than or equal)
 - < (less than)
 - <= (less than or equal)
 - = (equal)
 - <> (not equal)

- Results derived from comparisons can be zero (indicating `FALSE`) or non-zero (indicating `TRUE`).
- Parentheses (for grouping results in the expression)
- Conjunction operators: `AND`, `OR`

Use the following `FILTER` options to specify which SQL operations a filter clause affects. Any of these options can be combined.

`ON INSERT` | `ON UPDATE` | `ON DELETE IGNORE INSERT` | `IGNORE UPDATE` | `IGNORE DELETE`

Use the `RAISEERROR` option of `FILTER` in the `MAP` parameter to generate a user-defined error when the filter fails. This option is useful when you need to trigger an event in response to the failure.

You can use the `@RANGE` function to divide the processing workload among multiple `FILTER` clauses, using separate `TABLE` or `MAP` statements. For example, the following splits the replication workload into two ranges (between two Replicat processes or two threads of a coordinated Replicat) based on the `ID` column of the source `acct` table.

Table 11-6 Using Multiple `FILTER` Statements

Parameter file	Description
<code>REPERROR (9999, EXCEPTION)</code>	Raises an exception for the specified error.
<code>MAP OWNER.SRCTAB,</code> <code>TARGET OWNER.TARGETAB,</code>	Starts the <code>MAP</code> statement.
<code>SQLEXEC (ID CHECK, ON UPDATE,</code> <code>QUERY ' SELECT COUNT FROM TARGETAB '</code> <code>'WHERE PKCOL = :P1 ',</code> <code>PARAMS (P1 = PKCOL)),</code>	Performs a query to retrieve the present value of the <code>COUNT</code> column whenever an update is encountered.
<code>FILTER (BALANCE > 15000),</code>	Uses a <code>FILTER</code> clause to select rows where the balance is greater than 15000.
<code>FILTER (ON UPDATE, @BEFORE (COUNT) =</code> <code>CHECK.COUNT)</code>	Uses another <code>FILTER</code> clause to ensure that the value of the source <code>COUNT</code> column before an update matches the value in the target column before applying the target update.
<code>;</code>	The semicolon concludes the <code>MAP</code> statement.
<code>MAP OWNER.SRCTAB,</code> <code>TARGET OWNER.TARGETEXC,</code> <code>EXCEPTIONSONLY,</code> <code>COLMAP (USEDEFAULTS,</code> <code>ERRTYPE = 'UPDATE FILTER FAILED');</code>	Designates an exceptions <code>MAP</code> statement. The <code>REPERROR</code> clause for error 9999 ensures that the exceptions map to <code>TARGETEXC</code> will be executed.

Example 11-1 Calling the @COMPUTE Function

The following example calls the @COMPUTE function to extract records in which the price multiplied by the amount exceeds 10,000.

```
MAP SALES.TCUSTORD, TARGET SALES.TORD,
FILTER (@COMPUTE (PRODUCT_PRICE * PRODUCT_AMOUNT) > 10000);
```

Example 11-2 Calling the @STREQ Function

The following uses the @STREQ function to extract records where the value of a character column is 'JOE'.

```
TABLE ACCT.TCUSTORD, FILTER (@STREQ ("Name", 'joe') > 0);
```

Example 11-3 Selecting Records

The following selects records in which the AMOUNT column is greater than 50 and executes the filter on UPDATE and DELETE operations.

```
TABLE ACT.TCUSTORD, FILTER (ON UPDATE, ON DELETE, AMOUNT > 50);
```

Example 11-4 Using the @RANGE Function

(Replicat group 1 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (1, 2, ID));
```

(Replicat group 2 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (2, 2, ID));
```

You can combine several FILTER clauses in one MAP or TABLE statement, as shown in [Table 11-6](#), which shows part of a Replicat parameter file. Oracle GoldenGate executes the filters in the order listed, until one fails or until all are passed. If one filter fails, they all fail.

Selecting Rows with a WHERE Clause

Use any of the elements in [Table 11-7](#) in a WHERE clause to select or exclude rows (or both) based on a conditional statement. Each WHERE clause must be enclosed within parentheses. Literals must be enclosed within single quotes.

Table 11-7 Permissible WHERE Operators

Element	Examples
Column names	PRODUCT_AMT
Numeric values	-123, 5500.123
Literal strings	'AUTO', 'Ca'
Built-in column tests	@NULL, @PRESENT, @ABSENT (column is null, present or absent in the row). These tests are built into Oracle GoldenGate. See Considerations for Selecting Rows with FILTER and WHERE .

Table 11-7 (Cont.) Permissible WHERE Operators

Element	Examples
Comparison operators	=, <>, >, <, >=, <=
Conjunctive operators	AND, OR
Grouping parentheses	Use open and close parentheses () for logical grouping of multiple elements.

Oracle GoldenGate does not support `FILTER` for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.

Arithmetic operators and floating-point data types are not supported by `WHERE`. To use more complex selection conditions, use a `FILTER` clause or a user exit routine. See [Using User Exits to Extend Oracle GoldenGate Capabilities](#) for more information.

The syntax for `WHERE` is identical in the `TABLE` and `MAP` statements:

```
TABLE table, WHERE (clause);
```

```
MAP source_table, TARGET target_table, WHERE (clause);
```

Considerations for Selecting Rows with `FILTER` and `WHERE`

The following suggestions can help you create a successful selection clause.



Note:

The examples in this section assume a case-insensitive database.

- [Ensuring Data Availability for Filters](#)
- [Comparing Column Values](#)
- [Testing for NULL Values](#)

Ensuring Data Availability for Filters

If the database only logs values for *changed* columns to the transaction log, there can be errors if any of the unchanged columns are referenced by selection criteria. Oracle GoldenGate ignores such row operations, outputs them to the discard file, and issues a warning.

To avoid missing-column errors, create your selection conditions as follows:

- Use only primary-key columns as selection criteria, if possible.
- Make required column values available by enabling supplemental logging for those columns. Alternatively, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter. These options are valid for all supported databases. They query the database to fetch the values if they are not present in the log. To retrieve

the values before the `FILTER` or `WHERE` clause is executed, include the `FETCHBEFOREFILTER` option in the `TABLE` statement before the `FILTER` or `WHERE` clause. For example:

```
TABLE DEMO.PEOPLE, FETCHBEFOREFILTER, FETCHCOLS (age), FILTER (age > 50);
```

- Test for a column's presence first, then for the column's value. To test for a column's presence, use the following syntax.

```
column_name {= | <>} {@PRESENT | @ABSENT}
```

The following example returns all records when the `amount` column is over 10,000 and does not cause a record to be discarded when `amount` is absent.

```
WHERE (amount = @PRESENT AND amount > 10000)
```

Comparing Column Values

To ensure that elements used in a comparison match, compare appropriate column types:

- Character columns to literal strings.
- Numeric columns to numeric values, which can include a sign and decimal point.
- Date and time columns to literal strings, using the format in which the column is retrieved by the application.

Testing for NULL Values

To evaluate columns for `NULL` values, use the following syntax.

```
column {= | <>} @NULL
```

The following returns `TRUE` if the column is `NULL`, and `FALSE` for all other cases (including a column missing from the record).

```
WHERE (amount = @NULL)
```

The following returns `TRUE` only if the column is present in the record and not `NULL`.

```
WHERE (amount = @PRESENT AND amount <> @NULL)
```

Retrieving Before and After Values

For update operations, it can be advantageous to retrieve the *before* values of source columns: the values before the update occurred. These values are stored in the trail and can be used in filters and column mappings. For example, you can:

- Retrieve the before image of a row as part of a column-mapping specification in an exceptions `MAP` statement, and map those values to an exceptions table for use in testing or troubleshooting conflict resolution routines.
- Perform delta calculations. For example, if a table has a `Balance` column, you can calculate the net result of a particular transaction by subtracting the original balance from the new balance, as in the following example:

```
MAP "owner"."src", TARGET "owner"."targ",
COLMAP (PK1 = PK1, delta = balance - @BEFORE (balance));
```

 **Note:**

The previous example indicates a case-sensitive database such as Oracle. The table names are in quote marks to reflect case-sensitivity.

To Reference the Before Value

1. Use the `@BEFORE` column conversion function with the name of the column for which you want a before value, as follows:
`@BEFORE (column_name)`
2. Use the `GETUPDATEBEFORES` parameter in the Extract parameter file to capture before images from the transaction record, or use it in the Replicat parameter file to use the before image in a column mapping or filter. If using the Conflict Resolution and Detection (CDR) feature, you can use the `GETBEFORECOLS` option of `TABLE`. To use these parameters, all columns must be present in the transaction log. If the database only logs the values of columns that changed, using the `@BEFORE` function may result in a "column missing" condition and the column map is executed as if the column were not in the record. See [Ensuring Data Availability for Filters](#) to ensure that column values are available.

Oracle GoldenGate also provides the `@AFTER` function to retrieve after values when needed for filtering, for use in conversion functions, or other purposes. For more information about `@BEFORE` and `@AFTER`, see *Reference for Oracle GoldenGate*.

Selecting Columns

To control which columns of a source table are extracted by Oracle GoldenGate, use the `COLS` and `COLSEXCEPT` options of the `TABLE` parameter. Use `COLS` to select columns for extraction, and use `COLSEXCEPT` to select all columns except those designated by `COLSEXCEPT`.

Restricting the columns that are extracted can be useful when a target table does not contain the same columns as the source table, or when the columns contain sensitive information, such as a personal identification number or other proprietary business information.

Selecting and Converting SQL Operations

By default, Oracle GoldenGate captures and applies `INSERT`, `UPDATE`, and `DELETE` operations. You can use the following parameters in the Extract or Replicat parameter file to control which kind of operations are processed, such as only inserts or only inserts and updates.

`GETINSERTS` | `IGNOREINSERTS`

`GETUPDATES` | `IGNOREUPDATES`

`GETDELETES` | `IGNOREDELETES`

You can convert one type of SQL operation to another by using the following parameters in the Replicat parameter file:

- Use `INSERTUPDATES` to convert source update operations to inserts into the target table. This is useful for maintaining a transaction history on that table. The transaction log record must contain all of the column values of the table, not just changed values. Some databases do not log full row values to their transaction log, but only values that changed.
- Use `INSERTDELETES` to convert all source delete operations to inserts into the target table. This is useful for retaining a history of all records that were ever in the source database.
- Use `UPDATEDELETES` to convert source deletes to updates on the target.

Using Transaction History

Oracle GoldenGate enables you to retain a history of changes made to a target record and to map information about the operation that caused each change. This history can be useful for creating a transaction-based reporting system that contains a separate record for every operation performed on a table, as opposed to containing only the most recent version of each record.

For example, the following series of operations made to a target table named `CUSTOMER` would leave no trace of the ID of `Dave`. The last operation deletes the record, so there is no way to find out Dave's account history or his ending balance.

Table 11-8 Operation History for Table CUSTOMER

Sequence	Operation	ID	BALANCE
1	Insert	Dave	1000
2	Update	Dave	900
3	Update	Dave	1250
4	Delete	Dave	1250

Retaining this history as a series of records can be useful in many ways. For example, you can generate the net effect of transactions.

To Implement Transaction Reporting

1. To prepare Extract to capture before values, use the `GETUPDATEBEFORES` parameter in the Extract parameter file. A before value (or before image) is the existing value of a column before an update is performed. Before images enable Oracle GoldenGate to create the transaction record.
2. To prepare Replicat to post all operations as inserts, use the `INSERTALLRECORDS` parameter in the Replicat parameter file. Each operation on a table becomes a new record in that table.
3. To map the transaction history, use the return values of the `GGHEADER` option of the `@GETENV` column conversion function. Include the conversion function as the source expression in a `COLMAP` statement in the `TABLE` or `MAP` parameter.

Using the sample series of transactions shown in [Table 11-8](#) the following parameter configurations can be created to generate a more transaction-oriented view of customers, rather than the latest state of the database.

Process	Parameter statements
Extract	<pre>GETUPDATEBEFORES TABLE ACCOUNT.CUSTOMER;</pre>
Replicat	<pre>INSERTALLRECORDS MAP SALES.CUSTOMER, TARGET SALES.CUSTHIST, COLMAP (TS = @GETENV ('GGHEADER', 'COMMITTIMESTAMP'), BEFORE_AFTER = @GETENV ('GGHEADER', 'BEFOREAFTERINDICATOR'), OP_TYPE = @GETENV ('GGHEADER', 'OPTYPE'), ID = ID, BALANCE = BALANCE);</pre>

**Note:**

This is not representative of a complete parameter file for an Oracle GoldenGate process. Also note that these examples represent a case-insensitive database.

This configuration makes possible queries such as the following, which returns the net sum of each transaction along with the time of the transaction and the customer ID.

```
SELECT AFTER.ID, AFTER.TS, AFTER.BALANCE - BEFORE.BALANCE
FROM CUSTHIST AFTER, CUSTHIST BEFORE
WHERE AFTER.ID = BEFORE.ID AND AFTER.TS = BEFORE.TS AND
AFTER.BEFORE_AFTER = 'A' AND BEFORE.BEFORE_AFTER = 'B';
```

Testing and Transforming Data

Data testing and transformation can be performed by either Extract or Replicat and is implemented by using the Oracle GoldenGate built-in column-conversion functions within a COLMAP clause of a TABLE or MAP statement. With these conversion functions, you can:

- Transform dates.
- Test for the presence of column values.
- Perform arithmetic operations.
- Manipulate numbers and character strings.
- Handle null, invalid, and missing data.
- Perform tests.

This chapter provides an overview of some of the Oracle GoldenGate functions related to data manipulation. For the complete reference, see Reference for Oracle GoldenGate for Windows and UNIX.

If you need to use logic beyond that which is supplied by the Oracle GoldenGate functions, you can call your own functions by implementing Oracle GoldenGate user exits. See [Using User Exits to Extend Oracle GoldenGate Capabilities](#) for more information about user exits.

Oracle GoldenGate conversion functions take the following general syntax:

Syntax

@function (argument)

Table 11-9 Conversion Function Syntax

Syntax element	Description
<i>@function</i>	The Oracle GoldenGate function name. Function names have the prefix @, as in @COMPUTE or @DATE. A space between the function name and the open-parenthesis before the input argument is optional.
<i>argument</i>	A function argument.

Table 11-10 Function Arguments

Argument element	Example
A numeric constant	123
A string literal enclosed within single quote marks	'ABCD'
The name of a source column	PHONE_NO or phone_no, or "Phone_No" or Phone_no Depends on whether the database is case-insensitive, is case-sensitive and requires quote marks to enforce the case, or is case-sensitive and does not require quotes.
An arithmetic expression	COL2 * 100
A comparison expression	((COL3 > 100) AND (COL4 > 0))
Other Oracle GoldenGate functions	AMOUNT = @IF (@COLTEST (AMT, MISSING, INVALID), 0, AMT)

- [Handling Column Names and Literals in Functions](#)
- [Using the Appropriate Function](#)
- [Transforming Dates](#)
- [Performing Arithmetic Operations](#)
- [Manipulating Numbers and Character Strings](#)
- [Handling Null, Invalid, and Missing Data](#)
- [Performing Tests](#)

Handling Column Names and Literals in Functions

By default, literal strings must be enclosed in single quotes in a column-conversion function. Case-sensitive column names must be enclosed within double quotes if required by the database, or otherwise entered in the case in which they are stored in the database.

Using the Appropriate Function

Use the appropriate function for the type of column that is being manipulated or evaluated. For example, numeric functions can be used only to compare numeric values. To compare character values, use one of the Oracle GoldenGate character-comparison functions. LOB columns cannot be used in conversion functions.

This statement would fail because it uses `@IF`, which is a numerical function, to compare string values.

```
@IF (SR_AREA = 'Help Desk', 'TRUE', 'FALSE')
```

The following statement would succeed because it compares a numeric value.

```
@IF (SR_AREA = 20, 'TRUE', 'FALSE')
```

See [Manipulating Numbers and Character Strings](#) for more information.



Note:

Errors in argument parsing sometimes are not detected until records are processed. Verify syntax before starting processes.

Transforming Dates

Use the `@DATE`, `@DATEDIF`, and `@DATENOW` functions to retrieve dates and times, perform computations on them, and convert them.

This example computes the time that an order is filled

Example 11-5 Computing Time

```
ORDER_FILLED = @DATE (  
    'YYYY-MM-DD HH:MI:SS',  
    'JTS',  
    @DATE ('JTS',  
    'YYMMDDHHMISS',  
    ORDER_TAKEN_TIME) +  
    ORDER_MINUTES * 60 * 1000000)
```

Performing Arithmetic Operations

To return the result of an arithmetic expression, use the `@COMPUTE` function. The value returned from the function is in the form of a string. Arithmetic expressions can be combinations of the following elements.

- Numbers
- The names of columns that contain numbers
- Functions that return numbers
- Arithmetic operators:
 - + (plus)
 - - (minus)
 - * (multiply)
 - / (divide)
 - \ (remainder)
- Comparison operators:
 - > (greater than)
 - >= (greater than or equal)
 - < (less than)
 - <= (less than or equal)
 - = (equal)
 - <> (not equal)

Results that are derived from comparisons can be zero (indicating `FALSE`) or non-zero (indicating `TRUE`).

- Parentheses (for grouping results in the expression)
- The conjunction operators `AND`, `OR`. Oracle GoldenGate only evaluates the necessary part of a conjunction expression. Once a statement is `FALSE`, the rest of the expression is ignored. This can be valuable when evaluating fields that may be missing or null. For example, if the value of `COL1` is 25 and the value of `COL2` is 10, then the following are possible:

```
@COMPUTE ( (COL1 > 0) AND (COL2 < 3) ) returns 0.  
@COMPUTE ( (COL1 < 0) AND (COL2 < 3) ) returns 0. COL2 < 3 is never  
evaluated.  
@COMPUTE ((COL1 + COL2)/5) returns 7.
```

- [Omitting @COMPUTE](#)

Omitting @COMPUTE

The `@COMPUTE` keyword is not required when an expression is passed as a function argument.

```
@STRNUM ((AMOUNT1 + AMOUNT2), LEFT)
```

The following expression returns the same result as the previous one:

```
@STRNUM (@COMPUTE (AMOUNT1 + AMOUNT2), LEFT)
```

Manipulating Numbers and Character Strings

To convert numbers and character strings, Oracle GoldenGate supplies the following functions:

Table 11-11 Conversion Functions for Numbers and Characters

Purpose	Conversion Function
Convert a binary or character string to a number.	@NUMBIN
	@NUMSTR
Convert a number to a string.	@STRNUM
Compare strings.	@STRCMP
	@STRNCMP
Concatenate strings.	@STRCAT
	@STRNCAT
Extract from a string.	@STREXT
	@STRFIND
Return the length of a string.	@STRLEN
Substitute one string for another.	@STRSUB
Convert a string to upper case.	@STRUP
Trim leading or trailing spaces, or both.	@STRLTRIM
	@STRRTRIM
	@STRTRIM

Handling Null, Invalid, and Missing Data

When column data is missing, invalid, or null, an Oracle GoldenGate conversion function returns a corresponding value.

If `BALANCE` is 1000, but `AMOUNT` is `NULL`, the following expression returns `NULL`:

```
NEW_BALANCE = @COMPUTE (BALANCE + AMOUNT)
```

These exception conditions render the entire calculation invalid. To ensure a successful conversion, use the `@COLSTAT`, `@COLTEST` and `@IF` functions to test for, and override, the exception condition.

- [Using @COLSTAT](#)
- [Using @COLTEST](#)
- [Using @IF](#)

Using @COLSTAT

Use the `@COLSTAT` function to return an indicator to Extract or Replicat that a column is missing, null, or invalid. The indicator can be used as part of a larger manipulation formula that uses additional conversion functions.

The following example returns a `NULL` into target column `ITEM`.

```
ITEM = @COLSTAT (NULL)
```

The following `@IF` calculation uses `@COLSTAT` to return `NULL` to the target column if `PRICE` and `QUANTITY` are less than zero.

```
ORDER_TOTAL = PRICE * QUANTITY, @IF ((PRICE < 0) AND (QUANTITY < 0), @COLSTAT (NULL))
```

Using @COLTEST

Use the @COLTEST function to check for the following conditions:

- PRESENT tests whether a column is present and not null.
- NULL tests whether a column is present and null.
- MISSING tests whether a column is not present.
- INVALID tests whether a column is present but contains invalid data.

The following example checks whether the AMOUNT column is present and NULL and whether it is present but invalid.

```
@COLTEST (AMOUNT, NULL, INVALID)
```

Using @IF

Use the @IF function to return one of two values based on a condition. Use it with the @COLSTAT and @COLTEST functions to begin a conditional argument that tests for one or more exception conditions and then directs processing based on the results of the test.

```
NEW_BALANCE = @IF (@COLTEST (BALANCE, NULL, INVALID) OR  
@COLTEST (AMOUNT, NULL, INVALID), @COLSTAT (NULL), BALANCE + AMOUNT)
```

This conversion returns one of the following:

- NULL when BALANCE or AMOUNT is NULL or INVALID
- MISSING when either column is missing
- The sum of the columns.

Performing Tests

The @CASE, @VALONEOF, and @EVAL functions provide additional methods for performing tests on data before manipulating or mapping it.

- [Using @CASE](#)
- [Using @VALONEOF](#)
- [Using @EVAL](#)

Using @CASE

Use @CASE to select a value depending on a series of value tests.

```
@CASE (PRODUCT_CODE, 'CAR', 'A car', 'TRUCK', 'A truck')
```

This example returns the following:

- A car if PRODUCT_CODE is CAR
- A truck if PRODUCT_CODE is TRUCK
- A FIELD_MISSING indication if PRODUCT_CODE fits neither of the other conditions

Using @VALONEOF

Use @VALONEOF to compare a column or string to a list of values.

```
@IF (@VALONEOF (STATE, 'CA', 'NY'), 'COAST', 'MIDDLE')
```

In this example, if STATE is CA or NY, the expression returns COAST, which is the response returned by @IF when the value is non-zero (meaning TRUE).

Using @EVAL

Use @EVAL to select a value based on a series of independent conditional tests.

```
@EVAL (AMOUNT > 10000, 'high amount', AMOUNT > 5000, 'somewhat high')
```

This example returns the following:

- high amount if AMOUNT is greater than 10000
- somewhat high if AMOUNT is greater than 5000, and less than or equal to 10000, (unless the prior condition was satisfied)
- A FIELD_MISSING indication if neither condition is satisfied.

Using Tokens

You can capture and store data within the *user token* area of a trail record header. Token data can be retrieved and used in many ways to customize the way that Oracle GoldenGate delivers information. For example, you can use token data in:

- Column maps
- Stored procedures called by a `SQLEXEC` statement
- User exits
- Macros
- [Defining Tokens](#)
- [Using Token Data in Target Tables](#)

Defining Tokens

To use tokens, you define the token name and associate it with data. The data can be any valid character data or values retrieved from Oracle GoldenGate column-conversion functions.

The token area in the record header permits up to 16,000 bytes of data. Token names, the length of the data, and the data itself must fit into that space.

To define a token, use the `TOKENS` option of the `TABLE` parameter in the Extract parameter file.

Syntax

```
TABLE table_spec, TOKENS (token_name = token_data [, ...]);
```

Where:

- *table_spec* is the name of the source table. A container or catalog name, if applicable, and an owner name must precede the table name.
- *token_name* is a name of your choice for the token. It can be any number of alphanumeric characters and is not case-sensitive.
- *token_data* is a character string of up to 2000 bytes. The data can be either a string that is enclosed within single quotes or the result of an Oracle GoldenGate column-conversion function. The character set of token data is not converted. The token must be in the character set of the source database for Extract and in the character set of the target database for Replicat. In the trail file, user tokens are stored in UTF-8.

```
TABLE ora.oratest, TOKENS (
TK-OSUSER = @GETENV ('GGENVIRONMENT' , 'OSUSERNAME'),
TK-GROUP = @GETENV ('GGENVIRONMENT' , 'GROUPNAME')
TK-HOST = @GETENV('GGENVIRONMENT' , 'HOSTNAME'));
```

As shown in this example, the Oracle GoldenGate @GETENV function is an effective way to populate token data. This function provides several options for capturing environment information that can be mapped to tokens and then used on the target system for column mapping.

Using Token Data in Target Tables

To map token data to a target table, use the @TOKEN column-conversion function in the source expression of a COLMAP clause in a Replicat MAP statement. The @TOKEN function provides the name of the token to map. The COLMAP syntax with @TOKEN is:

Syntax

```
COLMAP (target_column = @TOKEN ('token_name'))
```

The following MAP statement maps target columns *host*, *gg_group*, and so forth to tokens *tk-host*, *tk-group*, and so forth. Note that the arguments must be enclosed within single quotes.

User tokens	Values
tk-host	:sysA
tk-group	:extora
tk-osuser	:jad
tk-domain	:admin
tk-ba_ind	:B
tk-commit_ts	:2011-01-24 17:08:59.000000

User tokens	Values
tk-pos	:3604496
tk-rba	:4058
tk-table	:oratest
tk-optype	:insert

Example 11-6 MAP Statement

```
MAP ora.oratest, TARGET ora.rpt,  
COLMAP (USEDEFAULTS,  
host = @token ('tk-host'),  
gg_group = @token ('tk-group'),  
osuser= @token ('tk-osuser'),  
domain = @token ('tk-domain'),  
ba_ind= @token ('tk-ba_ind'),  
commit_ts = @token ('tk-commit_ts'),  
pos = @token ('tk-pos'),  
rba = @token ('tk-rba'),  
tablename = @token ('tk-table'),  
optype = @token ('tk-optype'));
```

The tokens in this example will look similar to the following within the record header in the trail:

12

Associating Replicated Data with Metadata

This chapter describes the uses of metadata and how to associate replicated data with metadata.

Topics:

- [Overview](#)
- [Understanding Data Definition Files](#)
- [Using Automatic Trail File Recovery](#)
- [Configuring Oracle GoldenGate to Use Self-Describing Trail Files](#)
- [Configuring Oracle GoldenGate to Assume Identical Metadata](#)
- [Configuring Oracle GoldenGate to Assume Dissimilar Metadata](#)
- [Configuring Oracle GoldenGate to Use a Combination of Similar and Dissimilar Definitions](#)

Overview

When replicating data from one table to another, an important consideration is whether the column structures (metadata) of the source and target tables are identical. Oracle GoldenGate looks up metadata for the following purposes:

- On the source, to supply complete information about captured operations to the Replicat process.
- On the target, to determine the structures of the target tables, so that the replicated data is correctly mapped and converted (if needed) by Replicat.

In each of the following scenarios, you must use a different parameter or set of parameters to describe the metadata properly to the Oracle GoldenGate process that is processing it:

- You are replicating a source table to a target table that has identical metadata definitions (homogeneous replication).
- You are replicating a source table to a target table that has different metadata definitions.
- You are replicating a source table to two target tables, one with identical definitions and one that has different definitions.

Understanding Data Definition Files

Oracle GoldenGate can query the local database to get one set of definitions, but it must rely on a *data-definitions file* to get definitions from the remote database. The data-definitions file contains information about the metadata of the data that is being replicated. There are two types of definitions files:

- A *source-definitions file* contains the definitions of source tables.

- A *target-definitions file* contains the definitions of the target tables.

You can use multiple data-definitions files in a parameter file. For example, each one can contain the definitions for a distinct application.

- [Contents of the Definitions File](#)
- [Which Definitions File Type to Use, and Where](#)
- [Understanding the Effect of Character Sets on Definitions Files](#)
- [Using a Definitions Template](#)
- [Configuring Oracle GoldenGate to Capture Data-definitions](#)
- [Adding Tables that Satisfy a Definitions Template](#)
- [Examples of Using a Definitions File](#)

Contents of the Definitions File

The format of a data-definitions file is for internal use and should not be edited by an Oracle GoldenGate user unless instructed to do so in documented procedures or by a support representative. The file begins with a file header that shows the version of DEFGEN, information about character sets, the database type, the locale, and internal metadata that indicates other data properties. Following the header are the table-definition sections. Each table-definition section contains a table name, record length, number of columns, and one or more column definitions.

Which Definitions File Type to Use, and Where

The type of definitions file to use depends on where column mapping and conversion will be performed.

- When replicating from any type of Windows or UNIX-based database system to any other Windows or UNIX-based system, the mapping and conversion can be performed by Extract, a data-pump Extract, or Replicat, but is usually performed by Replicat on the target system. However, if Oracle GoldenGate must convert between different character sets, the mapping and conversion must be performed by Replicat on the target. See [Understanding the Effect of Character Sets on Definitions Files](#).
- When replicating from any Windows, UNIX, or Linux-based database system to an Enscribe target on a NonStop system, the mapping and conversion must be performed on the Windows, UNIX, or Linux system: Only Extract can convert two- and three-part SQL names and data types to the three-part file names that are used on the NonStop platform. In this scenario, Oracle GoldenGate cannot convert between source and target character sets. See [Understanding the Effect of Character Sets on Definitions Files](#).

Therefore:

- To perform column mapping and conversion on the target, use a *source-definitions file* that was generated on the source to supply the source definitions to Replicat.
- To perform column mapping and conversion on the source, use a *target-definitions file* that was generated on the target to supply target definitions to the primary Extract or a data-pump Extract, depending on which process does the conversion.
- To perform column mapping or transformation on an intermediary system, you may need to use multiple definition file types. See [Creating a Reporting](#)

[Configuration with a Data Pump on an Intermediary System](#) and [Creating a Cascading Reporting Configuration](#). Note that if there is not a Replicat on the intermediary system, conversion between character sets cannot be performed.

Understanding the Effect of Character Sets on Definitions Files

Oracle GoldenGate takes into consideration the character set encoding of the database when performing data conversion, and it takes into consideration the character set of the local operating system when creating a definitions file. Take the following guidelines into account when the source and target data have different character sets.

- [Confining Data Mapping and Conversion to the Replicat Process](#)
- [Avoiding File Corruptions Due to Operating System Character Sets](#)
- [Changing the Character Set of Existing Definitions Files](#)
- [Downloading from a z/OS system to another platform](#)

Confining Data Mapping and Conversion to the Replicat Process

Replicat is the only process that converts replicated data between different character sets. It converts data *from* the source database character set *to* the target database character set (or to the character set of the database on an intermediary system in a cascading configuration). As a result, data mapping and conversion must be performed by Replicat if source and target character sets are different. It cannot be performed on a source system, nor on an intermediary system that only contains a data pump. A target-definitions file is invalid in these cases.

Avoiding File Corruptions Due to Operating System Character Sets

By default, DEFGEN writes the definitions file itself in the character set of the local operating system. A definitions file can be created on the local system and transferred to the remote system without any encoding-related problems if the following is true:

- The remote system to which you are transferring the definitions file has the same or equivalent operating-system character set as the local system
- The operating-system character set of the remote system is a subset of the operating-system character set of the local system. For example, if the source and target character sets both are ASCII-compatible or EBCDIC-compatible and all table and column names use only 7-bit US-ASCII or equivalent characters, you can move the definition file between those systems.

Many operating-system character sets have little or no compatibility between them. To write the definitions file in a character set that is compatible with, or the same as, the one used by the remote system, use the `CHARSET` option of the `DEFSEFILE` parameter when you configure DEFGEN.

Changing the Character Set of Existing Definitions Files

In the case of an existing definitions file that is transferred to an operating system with an incompatible character set, you can run the DEFGEN utility on that system to convert the character set of the file to the required one. This procedure takes two input arguments: the name of the definitions file and the `UPDATECS character_set` parameter. For example:

```
defgen ./dirdef/source.def UPDATECS UTF-8
```

UPDATECS helps in situations such as when a Japanese table name on Japanese Windows is written in Windows CP932 to the data-definitions file, and then the definitions file is transferred to Japanese UNIX. The file cannot be used unless the UNIX is configured in PCK locale. Thus, you must use UPDATECS to convert the encoding of the definitions file to the correct format.

Downloading from a z/OS system to another platform

Definitions files generated on an IBM z/OS platform must be downloaded in BINARY mode when transferring them to a non-z/OS platform.

Using a Definitions Template

When you create a definitions file, you can specify a definitions template that reduces the need to create new definitions files when tables are added to the Oracle GoldenGate configuration after the initial startup. To use a template, all of the new tables must have identical structures, such as in a customer database where there are separate but identical tables for each customer (see [Rules for Tables to be Considered Identical](#)).

If you do not use a template and new tables are added after startup, you must generate a definitions file for each new table that is added to the Oracle GoldenGate configuration, then copy their contents to the existing master definitions file, and then restart the process.

Configuring Oracle GoldenGate to Capture Data-definitions

To configure Oracle GoldenGate to use a data-definitions file and template (if needed), you will:

Topics:

- [Configure DEFGEN](#)
- [Run DEFGEN](#)
- [Transfer the Definitions File to the Remote System](#)
- [Specify the Definitions File](#)

Configure DEFGEN

Perform these steps on the system from which you want to obtain metadata definitions.



Note:

Do not create a data-definitions file for Oracle sequences. It is not needed and DEFGEN does not support it.

1. From the Oracle GoldenGate directory, run GGSCI.

2. In GGSCI, issue the following command to create a DEFGEN parameter file.

```
EDIT PARAMS DEFGEN
```
3. Enter the parameters listed in [Table 12-1](#) in the order shown, starting a new line for each parameter statement.

Table 12-1 DEFGEN Parameters

Parameter	Description
<code>CHARSET <i>character_set</i></code>	Use this parameter to specify a character set that DEFGEN will use to read the parameter file. By default, the character set of the parameter file is that of the local operating system. If used, CHARSET must be the first line of the parameter file.
<code>DEFSSFILE <i>file_name</i> [APPEND PURGE] [CHARSET <i>character_set</i>] [FORMAT RELEASE <i>major.minor</i>]</code>	<p>Specifies the relative or fully qualified name of the data-definitions file that is to be the output of DEFGEN.</p> <p>See <i>Reference for Oracle GoldenGate</i> for important information about these parameter options and their effect on character sets.</p> <p>See Understanding the Effect of Character Sets on Definitions Files for more information.</p> <ul style="list-style-type: none"> • APPEND directs DEFGEN to write new content (from the current run) at the end of any existing content, if the specified file already exists. • PURGE directs DEFGEN to purge the specified file before writing new content from the current run. This is the default. • CHARSET generates the definitions file in the specified character set instead of the default character set of the operating system. • FORMAT RELEASE specifies the Oracle GoldenGate release version of the definitions file. Use when the definitions file will be read by a process that is in an earlier version of Oracle GoldenGate than the DEFGEN process.
<code>[{SOURCEDB TARGETDB} <i>datasource</i>] {USERIDALIAS <i>alias</i> USERID <i>user</i>, PASSWORD <i>password</i> [<i>encryption_options</i>]}</code>	<p>Specifies database connection information.</p> <p>The <i>datasource</i> can be a DSN (Datasource Name), or a container of an Oracle container database (CDB). If connecting to an Oracle container database, connect to the root container as the common user if you need to generate definitions for objects in more than one container. Otherwise, you can connect to a specific container to generate definitions only for that container.</p> <p>For more information about SOURCEDB, USERID, and USERIDALIAS, including the databases they support, see <i>Reference for Oracle GoldenGate</i>.</p> <ul style="list-style-type: none"> • SOURCEDB TARGETDB specifies a data source name, if required as part of the connection information. Not required for Oracle. • USERID <i>user</i>, PASSWORD <i>password</i> [<i>encryption_options</i>] specifies a user name and password, with optional encryption options. • USERIDALIAS supplies database authentication through credentials stored in the Oracle GoldenGate credential store.
<code>NOCATALOG</code>	Removes the container name (Oracle) from table names before their definitions are written to the definitions file. Use this parameter if the definitions file is to be used for mapping to a database that only supports two-part names (<i>owner.object</i>).

Table 12-1 (Cont.) DEFGEN Parameters

Parameter	Description
<p>TABLE <i>container.owner.table</i> [, {DEF TARGETDEF} <i>template</i>];</p> <p>Where:</p> <ul style="list-style-type: none"> <i>container</i> is a container in an Oracle container database. <i>owner</i> is the name of the schema that contains the table to be defined. <i>table</i> is the table that is to be defined. [, {DEF TARGETDEF} <i>template</i>] additionally creates a definitions template based on the metadata of this table. This option is not supported for initial loads. See <i>Reference for Oracle GoldenGate</i> for information about this option. 	<p>Specifies the fully qualified name of a table or tables for which definitions will be defined and optionally uses the metadata of the table as a basis for a definitions template. Case-sensitivity of both table name and template name is preserved for case-sensitive databases. See Specifying Object Names in Oracle GoldenGate Input for instructions on wildcarding and case-sensitivity.</p> <p>Specify a source table(s) if generating a source-definitions file or a target table(s) if generating a target-definitions file.</p> <p>To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.</p> <p>Note that DEFGEN does not support UDTs.</p>

4. Save and close the file.
5. Exit GGSCI.

Run DEFGEN

1. From the directory where Oracle GoldenGate is installed, run DEFGEN using the following arguments. This example shows a UNIX file system structure.

```
defgen paramfile dirprm/defgen.prm [reportfile dirrpt/defgen.rpt]
[NOEXTATTR]
```

Where:

- `defgen` is the name of the program.
 - `paramfile` is a required keyword. `dirprm/defgen.prm` is the relative or full path name of the DEFGEN parameter file. (The typical location is shown in the example.)
 - `reportfile` is a required keyword. `dirrpt/defgen.rpt` sends output to the screen and to the designated report file. (The typical location is shown in the example.) You can omit the `reportfile` argument to print to the screen only.
 - `NOEXTATTR` can be used to support backward compatibility with Oracle GoldenGate versions that are older than Release 11.2.1 and do not support character sets other than ASCII, nor case-sensitivity or object names that are quoted with spaces. `NOEXTATTR` prevents DEFGEN from including the database locale and character set that support the globalization features that were introduced in Oracle GoldenGate Release 11.2.1. If the table or column name has multi-byte or special characters such as white spaces, DEFGEN does not include the table definition when `NOEXTATTR` is specified. If `APPEND` mode is used in the parameter file, `NOEXTATTR` is ignored, and the new table definition is appended in the existing file format, whether with the extra attributes or not.
2. Repeat these steps for any additional definitions files that you want to create.
 3. Using ASCII mode, FTP the definitions file (or files) from the local Oracle GoldenGate `dirdef` sub-directory to the remote `dirdef` sub-directory.

Transfer the Definitions File to the Remote System

Use `BINARY` mode to FTP the data definitions file to the remote system if the local and remote operating systems are different and the definitions file is created for the remote operating system character set. This avoids unexpected characters to be placed in the file by the FTP program, such as new-line and line-feed characters. Always use `BINARY` mode when transferring definitions files from z/OS to a non-z/OS platform.

Specify the Definitions File

Associate a data-definitions file with the correct Oracle GoldenGate process in the following ways:

- Associate a target-definitions file with an Extract group or data pump by using the `TARGETDEFS` parameter in the Extract parameter file.
- Associate a source-definitions file with the Replicat group by using the `SOURCEDEFS` parameter in the Replicat parameter file.
- If Oracle GoldenGate is to perform mapping or conversion on an intermediary system that contains neither the source nor target database, associate a source-definitions file and a target-definitions file with the data pump Extract by using `SOURCEDEFS` and `TARGETDEFS` in the parameter file. For Oracle databases, the Oracle libraries also must be present on the intermediary system.

See [Examples of Using a Definitions File](#) for the correct way to specify multiple definitions files.

Do not use `SOURCEDEFS` and `ASSUMETARGETDEFS` in the same parameter file. See [Configuring Oracle GoldenGate to Assume Identical Metadata](#) for more information about `ASSUMETARGETDEFS`.

Adding Tables that Satisfy a Definitions Template

To map a new table in the Oracle GoldenGate configuration to a definitions template, use the following options of the `TABLE` and `MAP` parameters, as appropriate:

- `DEF` to specify the name of a source-definitions template.
- `TARGETDEF` to specify the name of a target-definitions template.

Because these options direct the Extract or Replicat process to use the same definitions as the specified template, you need not create a new definitions file for the new table, nor restart the process.

Examples of Using a Definitions File

This topic contains some basic use cases that include a definitions file.

Topics:

- [Creating a Source-definitions file for Use on a Target System](#)
- [Creating Target-definitions Files for Use on a Source System](#)
- [Creating Multiple Source Definition Files for Use on a Target System](#)

Creating a Source-definitions file for Use on a Target System

The following configuration uses a DEFGEN parameter file that creates a source-definitions file as output. This example is for tables from an Oracle database.

```
DEFSFILE C:\ggs\dirdef\record.def
USERIDALIAS ogg
TABLE acct.cust100, DEF custdef;
TABLE ord.*;
TABLE hr.*;
```

The results of this DEFGEN configuration are:

- Individual definitions by name are created for all tables in the `ord` and `hr` schemas.
- A `custdef` template is created based on table `acct.cust100`. In the database, there are other `acct.cust*` tables, each with identical definitions to `acct.cust100`.

The tables are mapped in the Replicat parameter file as follows:

```
-- This is a simplified parameter file. Your requirements may vary.
REPLICAT acctrep
USERIDALIAS ogg
SOURCEDEFS c:\ggs\dirdef\record.def
MAP acct.cust*, TARGET acct.cust*, DEF custdef;
MAP ord.prod, TARGET ord.prod;
MAP ord.parts, TARGET ord.parts;
MAP hr.emp, TARGET hr.emp;
MAP hr.salary, TARGET hr.salary;
```

Note that definitions for tables that satisfy the wildcard specification `acct.cust*` are obtained from the `custdef` template, as directed by the `DEF` option of the first `MAP` statement.

Creating Target-definitions Files for Use on a Source System

If target definitions are required for the same tables, those tables can be mapped for a primary Extract or a data pump.

- Target definitions are required instead of source definitions if the target is an Enscribe database.
- Target definitions are required in addition to source definitions if mapping and conversion are to be done on an intermediary system.

The DEFGEN configuration to make the target-definitions file looks similar to the following:

```
DEFSFILE C:\ggs\dirdef\trecord.def
USERIDALIAS ogg
TABLE acct.cust100, DEF tcustdef;
TABLE ord.*;
TABLE hr.*;
```

 **Note:**

See the previous example for the DEFGEN configuration that makes the source-definitions file.

The Extract configuration looks similar to the following:

```
-- This is a simplified parameter file. Your requirements may vary.
EXTRACT acctex
USERIDALIAS ogg
RMTHOSTOPTIONS sysb, MGRPORA 7890, ENCRYPT AES192 KEYNAME mykey1
ENCRYPTTRAIL AES192
RMTTRAIL $data.ggsdat.rt
SOURCEDEFS c:\ggs\dirdef\record.def
TARGETDEFS c:\ggs\dirdef\trecord.def
TABLE acct.cust*, TARGET acct.cust*, DEF custdef, TARGETDEF tcustdef;
TABLE ord.prod, TARGET ord.prod;
TABLE ord.parts, TARGET ord.parts;
TABLE hr.emp, TARGET hr.emp;
TABLE hr.salary, TARGET hr.salary;
```

In this example, the source template named `custdef` (from the `record.def` file) and a target template named `tcustdef` (from the `trecord.def` file) are used for the `acct.cust*` tables. Definitions for the tables from the `ord` and `hr` schemas are obtained from explicit definitions based on the table names (but a wildcard specification could have been used here, instead)

Creating Multiple Source Definition Files for Use on a Target System

This is a simple example of how to use multiple definitions files. Your parameter requirements may vary, based on the Oracle GoldenGate topology and database type.

The following is the DEFGEN parameter file that creates the first data-definitions file.

```
DEFSSFILE C:\ggs\dirdef\sales.def
USERIDALIAS ogg
TABLE ord.*;
```

The following is the DEFGEN parameter file that creates the second data-definitions file. Note the file name and table specification are different from the first one.

```
DEFSSFILE C:\ggs\dirdef\admin.def
USERIDALIAS ogg
TABLE hr.*;
```

The tables for the first and second definitions file are mapped in the same Replicat parameter file as follows:

```
REPLICAT acctrep
USERIDALIAS ogg
SOURCEDEFS c:\ggs\dirdef\sales.def
MAP ord.*, TARGET ord.*;
SOURCEDEFS c:\ggs\dirdef\admin.def
MAP hr.*, TARGET hr.*;
```

Using Automatic Trail File Recovery

The trail recovery process has the ability to, in some cases, automatically rebuild trail files that are corrupt or missing by Oracle GoldenGate. When an Extract pump restarts, if the last trail that the pump was writing to is missing, then the Extract pump attempts to rebuild the missing trail file on the target system. This is done automatically using the checkpoint information for the process and the last valid trail file. The Replicat process automatically skips over any duplicate data in the trail files that have been rebuilt by the new trail recovery feature. This recovery will occur as long as there is at least 1 target trail from this sequence and that the trail files still exist on the source where the Extract pump is reading them.

This process can also be used to rebuild corrupt or invalid trail files on the target. Simply delete the corrupt trail file, and any trail files after that, and then restart the Extract pump. With this new behavior, Oracle recommends that `PURGEOLDEXTRACTS` `MINKEEP` rules are properly configured to ensure that there are trail files from the source that can be used to rebuild the target environment. This feature requires that Oracle GoldenGate release 12.1.2.1.8 or greater is used on both the source and target servers. Do *not* attempt to start the Replicat with `NOFILTERDUPTRANSACTIONS` because it will override Replicat's default behavior and may cause transactions that have already been applied to the target database to be applied again.

Configuring Oracle GoldenGate to Use Self-Describing Trail Files

The default behavior in this release is to store and forward metadata from the source to the target and encapsulates it in each of the trail files. In other words, a self-describing Extract trail or file is created by adding the metadata records in each file. There are two types of metadata records:

- *Database Definition Record (DDR)*

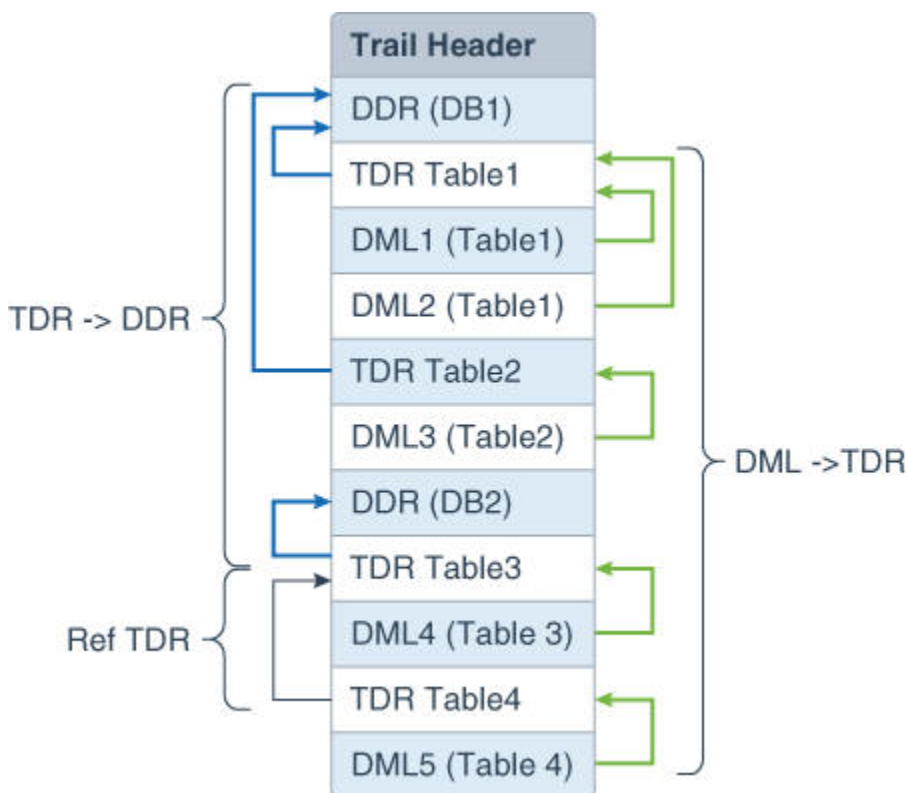
A DDR provides information about a specific database, such as character set and time zone. A Database Definition Record is added to the trail to store the database metadata for each pluggable database being captured. All the row change records from a pluggable database will have character and timestamp with local time zone data based on the corresponding DDR for that pluggable database. DDRs are generated for both consolidated and non-consolidated databases.

- *Table Definition Record (TDR)*

A TDR provides details about the definition about a table and the columns that it contains. The content of this record is similar to what is provided in a definitions file, which is a subset of the information found in the `file_def` and `col_def` classes. Each database can embed its own database specific information to each TDR. A TDR contains a complete table definition and is used to describe many row change records for the same table. A new TDR is written when the output trail rolls over to a new file or the source table definition has changed.

It is important to note that a TDR describes the definition of a table object represented by the row change records. It will be similar though may not be identical to the table definition in the source. For example, if a column-conversion function is applied to a source column, the metadata for that value in the database will be different from the metadata that shows up in a trail file.

The metadata records in a self-describing trail file format operate as follows:



Using self-described trail files eliminates the need for `SOURCEDEFS` and `ASSUMETARGETDEFS` so parameter files are simpler and it is simpler to configure heterogeneous replication and provides:

- A reduction in trail file size due to object name compression.
- The ability to extract data from multiple catalogs with different character sets and time zones into one trail.
- The ability to configure DDL replication among more than two Oracle databases. There is no need to use the `GETREPLICATS`, `UPDATEMETADATA`, and `NOTAG` parameters. You can replicate DDLs when source and target tables are not alike and without having to synchronize Oracle GoldenGate processes .
- No necessity to create and maintain source definitions files.

Understanding the Self-Describing Trail Behavior

When you are modifying the Source Table Definition the following criteria must be met to update the new TDR into the Extract's memory, as well as the trail file.

Oracle Database Sources

Integrated Extract (with Oracle Database 11.2.0.4 or higher and `compatible = 11.2.0.4` or higher): No manual steps are needed because Integrated Extract seamlessly generates updated metadata records after a DDL is performed on the source table. This is true irrespective of whether DDL replication is enabled or not.

Classic Extract: The Extract parameter file should include DDL parameter for Extract to seamlessly generate updated metadata records after a DDL. Alternatively, as in-

releases earlier than 12.2 , DDL should be performed only after Extract has completely output all the relevant database changes to the trail and is stopped. After DDL is executed, you must restart the Extract. Unlike previous releases, there is no need to stop Replicat and regenerate `SOURCEDEFS` using `DEFGEN`.

Heterogeneous Database Sources

As in releases earlier than 12.2, DDL should be performed only after Extract has completely output all the relevant database changes to the trail and is stopped. After DDL is executed, you must restart the Extract should be restarted. Unlike previous releases, there is no need to stop Replicat and regenerate `SOURCEDEFS` using `DEFGEN`.

Topics:

- [Support Considerations](#)
- [Using Self-Describing Trail Files](#)
- [Examples of Parameter Files](#)

Support Considerations

Review the following support information:

- Trail File Formats:
 - Must be Oracle GoldenGate release 12c (12.2.0.1) or greater to contain metadata records.
 - Cannot generate a 12c (12.2.0.1) trail format with the older trail format in a multi-trail configuration.
 - `FORMATASCII`, `FORMATSQL` and `FORMATXML` trails will not contain metadata records.
- For existing trail file configurations, you can easily switch between the previous and self-describing extract trail methods of resolving the table metadata by:
 - Use the `USE_TRAILDEFS GLOBALS` parameter to control all pumps and Replicats.
 - Use the `OVERRIDE` option of `SOURCEDEFS` and `ASSUMETARGETDEFS` to control an individual pump or Replicat. Oracle does not recommend this.
- Logdump displays the metadata records similar to `DEFGEN` output.
- Reverse is not supported in the 12c (12.2.0.1) trail format.
- If a table is mapped, the generated TDR is based on the definition of the mapped table not the source table.
- Metadata in the trail is supported for all databases except HP NonStop (Guardian).

Using Self-Describing Trail Files

Use the `USE_TRAILDEFS GLOBALS` parameter to enable or disable all pumps and Replicats. This command usage in relation to the `SOURCEDEFS` and `ASSUMETARGETDEF`, and its source table definitions are described as follows.

Figure 12-1 USE_TRAILDEFS | NOUSE_TRAILDEFS USAGE

		GLOBALS Parameter	
		USE_TRAILDEFS	NO_USE_TRAILDEFS
Extract/ Pump Parameter	SOURCEDEFS	Use the definitions from the trail. Issue a warning that SOURCEDEFS is ignored.	Use the definitions from the SOURCEDEFS file.
	SOURCEDEFS OVERRIDE	Use the definitions from the SOURCEDEFS file.	Use the definitions from the SOURCEDEFS file.
	ASSUMETARGETDEFS	Use the definitions from the trail. Issue a warning that ASSUMETARGETDEFS is ignored.	Use the definitions from the target database.
	ASSUMETARGETDEFS OVERRIDE	Use the definitions from the target database.	Use the definitions from the target database.

You must use the `OVERRIDE` option with the `ASSUMETARGETDEFS` and `SOURCEDEFS` parameters when using self-describing trail files.

Examples of Parameter Files

The following is an example of an Extract parameter file:

```
EXTRACT ext1
USERID tkggadmin@inst1, password tkggadmin
DDL include objname hr.*, include objname st_hr.*
RMTTRAIL $data/ggs12.2/a1
TABLE hr.*;
TABLE st_hr.salary, TARGET hr.salary, COLMAP (USEDEFAULTS,
                                             ts = @GETENV('GGHEADER' , 'COMMITTIMESTAMP'));
RMTTRAIL $data/ggs12.2/a2, NO_OBJECTDEFS
TABLE orders.*;
```

The following is an example of an Replicat parameter file:

```
REPLCAT repl
USERID tkggadmin@inst2, password tkggadmin
DDLERROR default discard
DDL include all
DISCARDFILE ./dirrpt/repl.dsc purge
MAP hr.*, TARGET hr.*;
```

Configuring Oracle GoldenGate to Assume Identical Metadata



Note:

This section does not apply to self-describing trail files.

When source and target tables have identical metadata definitions, use the `ASSUMETARGETDEFS` parameter in the Replicat parameter file. This parameter directs Replicat to assume that the target definitions are the same as those of the source, and to apply those definitions to the replicated data when constructing SQL statements. The source and target tables must be identical in every way, thus needing no conversion processing, although their catalogs or containers, owners and/or names can be different.

- [Rules for Tables to be Considered Identical](#)

Rules for Tables to be Considered Identical

For source and target structures to be identical, they must:

- be of the same database type, that is, all Oracle.
- have the same character set and locale, for example `american_AMERICA`.
- contain the same number of columns.
- have identical column names (including case, white spaces, and quotes if applicable).
- have identical data types.
- have identical column lengths.
- have the same column length semantics for character columns (bytes versus characters).
- define all of the columns in the same order.

The following is a simple Replicat parameter file that illustrates the use of `ASSUMETARGETDEFS`. For more information, see `ASSUMETARGETDEFS` in *Reference for Oracle GoldenGate*.

```
-- Specifies the group name.
REPLICAT acctrep
-- Specifies database login with an alias to a credential in the credential store.
USERIDALIAS ogg
-- Specifies a file for discard output.
DISCARDFILE ./dirrpt/backup/r_prod.dsc, APPEND
-- States that source and target definitions are identical.
ASSUMETARGETDEFS
-- Maps source tables to target tables.
MAP hq.product, TARGET region1.product;
MAP hq.price, TARGET region1.price;
```

When source and target structures are different, use the `SOURCEDEFS` parameter. See [Configuring Oracle GoldenGate to Assume Dissimilar Metadata](#). `ASSUMETARGETDEFS` and `SOURCEDEFS` cannot be used in the same parameter file.

Configuring Oracle GoldenGate to Assume Dissimilar Metadata

Source and target metadata definitions are not considered identical if they do not meet the rules in [Rules for Tables to be Considered Identical](#). When source and target table definitions are dissimilar, Oracle GoldenGate must perform a conversion from one format to the other. To perform conversions, both sets of definitions must be known to Oracle GoldenGate.

Configuring Oracle GoldenGate to Use a Combination of Similar and Dissimilar Definitions

Note:

This section does not apply to self-describing trail files.

`ASSUMETARGETDEFS` and `SOURCEDEFS` can be used in the same parameter file. This can be done when column mapping or conversion must be performed between some of the source-target table pairs, but not for other table pairs that are identical.

The following is an example of how to use `SOURCEDEFS` and `ASSUMETARGETDEFS` in the same parameter file. This example builds on the previous examples where tables in the `acct`, `ord`, and `hr` schemas require `SOURCEDEFS`, but it adds a `rpt` schema with tables that are dynamically created with the name `stock` appended with a random numerical value. For Oracle GoldenGate to replicate the DDL as well as the DML, the target tables must be identical. In that case, `ASSUMETARGETDEFS` is required.

```
REPLICAT acctrep
USERIDALIAS ogg
SOURCEDEFS c:\ggs\dirdef\record.def
MAP acct.cust*, TARGET acct.cust*, DEF custdef;
MAP ord.prod, TARGET ord.prod;
MAP ord.parts, TARGET ord.parts;
MAP hr.emp, TARGET hr.emp;
MAP hr.salary, TARGET hr.salary;
ASSUMETARGETDEFS
MAP rpt.stock, TARGET rpt.stock;
```


13

Configuring Online Change Synchronization

This chapter describes how to configure online change synchronization.

Topics:

- [Overview of Online Change Synchronization](#)
- [Choosing Names for Processes and Files](#)
- [Creating a Checkpoint Table](#)
- [Creating an Online Extract Group](#)
- [Creating a Trail](#)
- [Creating a Parameter File for Online Extraction](#)
- [Creating an Online Replicat Group](#)
- [Creating a Parameter File for Online Replication](#)

Overview of Online Change Synchronization

Online change synchronization extracts and replicates data changes continuously to maintain a near real-time target database. The number of Extract and Replicat processes and trails that you will need depends on the replication topology that you want to deploy and the process mode that you will be using.

For detailed information about deploying specific replication topologies, see:

- [Using Oracle GoldenGate for Live Reporting](#)
- [Using Oracle GoldenGate for Real-time Data Distribution](#)
- [Configuring Oracle GoldenGate to Maintain a Live Standby Database](#)
- [Configuring Oracle GoldenGate for Active-Active High Availability](#)

You may need to configure multiple Replicat processes if you are replicating between Oracle multitenant container databases.

You may need to configure multiple process groups to achieve a certain performance level. For example, you may want to keep lag below a certain threshold. Lag is the difference between when changes are made within your source applications and when those changes are applied to the target database.

Oracle GoldenGate supports up to 5,000 concurrent Extract and Replicat groups per instance of Oracle GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the `INFO` and `STATUS` commands. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at the default level of 300 or below in order to manage your environment effectively.

See [Tuning the Performance of Oracle GoldenGate](#) for more information about configuring Oracle GoldenGate for best performance.

- [Initial Synchronization](#)

Initial Synchronization

After you configure your change-synchronization groups and trails following the directions in this chapter, see [Instantiating Oracle GoldenGate with an Initial Load](#) to prepare the target tables for synchronization. An initial load takes a copy of entire source tables, transforms the data if necessary, and applies it to the target tables so that the movement of transaction data begins from a synchronized state. The first time that you start change synchronization should be during the initial synchronization process. Change synchronization keeps track of ongoing transactional changes while the load is being applied.

Choosing Names for Processes and Files

It is helpful to develop consistent naming conventions for the Oracle GoldenGate processes and files before you start configuration steps. Choosing meaningful names helps you differentiate among multiple processes and files in displays, error logs, and external monitoring programs. In addition, it accommodates the naming of additional processes and files later, as your environment changes or expands.

This section contains instructions for:

- [Naming Conventions for Processes](#)
- [Choosing File Names](#)

Naming Conventions for Processes

When specifying a process or group name, follow these rules.

- For the following types of processes, you can use up to eight characters, including non-alphanumeric characters such as the underscore (`_`):
 - Online Extract group
 - Initial-load Extract
 - Online Replicat group created in classic (non-coordinated) mode
 - Online Replicat group created in integrated mode (Oracle only)
- For coordinated and parallel Replicat process group, you can use up to five characters, including non-alphanumeric characters such as the underscore (`_`). Internally, a three-character thread ID is appended to the base name for each thread that is created based on the `MAXTHREADS` option of the `ADD REPLICAT` command. The resulting names cannot be duplicated for any other Replicat group. For example, if a coordinated Replicat group named `fin` is created with a `MAXTHREADS` of 50 threads, the resulting thread names could span from `fin000` through `fin050`, assuming those are the IDs specified in the `MAP` statements. Thus, no other Replicat group can be named `fin000` through `fin0050`. See the following rule for more information.
- You can include a number in a group name, but it is not recommended that a name end in any numerals. Understand that using a numeric value at the end of a

group name (such as `fin1`) can cause duplicate report file names and errors, because the writing process appends a number to the end of the group name when generating a report. In addition, ending a group name with numeric values is not recommended when running Replicat in coordinated mode. Because numeric thread IDs are appended to a group name internally, if the base group name also ends in a number it can make the output of informational commands more complicated to analyze. Thread names could be confused with the names of other Replicat groups if the numeric appendages satisfy wildcards. Duplicate report file names also can occur. It may be more practical to put a numeric value at the beginning of a group name, such as `1_fin`, `1fin`, and so forth.

- Any character can be used in the name of a process, so long as the character set of the local operating system supports it, and the operating system allows that character to be in a file name. This is because a group is identified by its associated checkpoint file and parameter file.
- The following characters are not allowed in the name of a process:
`{ \ / : * ? " < > | }`
- On HP UX, Linux, and Solaris, it is possible to create a file name with a colon (`:`) or an asterisk (`*`), although it is not recommended.
- In general, process names and parameter file names are not case-sensitive within Oracle GoldenGate. For example, `finance`, `Finance`, and `FINANCE` are all considered to be the same. However, on Linux, the process name (and its parameter file name if explicitly defined in the `ADD` command) must be all uppercase or all lowercase. Mixed-case names specified for processes and parameter files will result in errors when starting the process.
- Use only one word for a name.
- Do not use the word "port" as the full name for a process or parameter file. However, the string "port" can be part of a name.

Choosing File Names

Captured data must be processed into a series of files called a trail, where it is stored for processing by the next Oracle GoldenGate process downstream. The basic configuration is:

- A local trail on the source system
- A remote trail on the target system

The actual trail name contains only two characters, such as `./dirdat/tr`. Oracle GoldenGate appends this name with a nine-digit sequence number whenever a new file is created, such as `./dirdat/aa00000002`. It is recommended that you establish naming conventions for trails, because they are linked to Oracle GoldenGate processes and may need to be identified for the purposes of troubleshooting.

On Windows systems, if the name of any directory in the trail path name begins with a number, the path must be specified with forward slashes, not backward slashes, when listing the trail in a parameter file. For more information, see [Specifying Filesystem Path Names in Parameter Files on Windows Systems](#).

See [What is a Trail?](#) for more information about Oracle GoldenGate trails.

Creating a Checkpoint Table

Replicat maintains checkpoints that provide a known position in the trail from which to start after an expected or unexpected shutdown. To store a record of its checkpoints, Replicat uses a checkpoint table in the target database. This enables the Replicat checkpoint to be included within the Replicat transaction itself, to ensure that a transaction will only be applied once, even if there is a failure of the Replicat process or the database process. The checkpoint table remains small because rows are deleted when no longer needed, and it does not affect database performance. [About Checkpoints](#) for more information about the checkpoint table.

- [Options for Creating the Checkpoint Table](#)
- [Adjusting for Coordinated Replicat in Oracle RAC](#)

Options for Creating the Checkpoint Table

The checkpoint table can reside in a schema of your choice. Use one that is dedicated to Oracle GoldenGate if possible.

More than one instance of Oracle GoldenGate (multiple installations) can use the same checkpoint table. Oracle GoldenGate keeps track of the checkpoints, even if Replicat group names are the same in different instances.

More than one checkpoint table can be used as needed. For example, you can use different ones for different Replicat groups.

You can install your checkpoint tables in these ways:

- You can specify a default checkpoint table in the `GLOBALS` file. New Replicat groups created with the `ADD REPLICAT` command will use this table automatically, without requiring any special instructions. See ["To Specify a Default Checkpoint Table in the GLOBALS File"](#) for instructions.
- You can provide specific checkpoint table instructions when you create any given Replicat group with the `ADD REPLICAT` command:
 - To use a specific checkpoint table for a group, use the `CHECKPOINTTABLE` argument of `ADD REPLICAT`. This checkpoint table overrides any default specification in the `GLOBALS` file. If using only one Replicat group, you can use this command and skip creating the `GLOBALS` file altogether.
 - To omit using a checkpoint table for a group, use the `NODBCHECKPOINT` argument of `ADD REPLICAT`. Without a checkpoint table, Replicat still maintains checkpoints in a checkpoint file on disk, but you introduce the risk of data inconsistency.

However you implement the checkpoint table, you must create it in the target database prior to using the `ADD REPLICAT` command.

To Add a Checkpoint Table to the Target Database

The following steps, which create the checkpoint table through GGSCI, can be bypassed by running the `chkpt_db_create.sql` script instead, where `db` is an abbreviation of the database type. By using the script, you can specify custom storage or other attributes. Do not change the names or attributes of the columns in this table.

1. From the Oracle GoldenGate directory, run GGSCI and issue the `DBLOGIN` command to log into the database. The user issuing this command must have `CREATE TABLE` permissions. See *Reference for Oracle GoldenGate* for the correct syntax to use for your database.
2. In GGSCI, issue the following command to add the checkpoint table to the database.

```
ADD CHECKPOINTTABLE container owner.table
```

Where:

owner.table is the owner and name of the table, *container* is the name of a PDB if installing into an Oracle multitenant container database. The owner and name can be omitted if you are using this table as the default checkpoint table and this table is specified with `CHECKPOINTTABLE` in the `GLOBALS` file. The name of this table must not exceed the maximum length permitted by the database for object names. The checkpoint table name cannot contain any special characters, such as quotes, backslash, pound sign, and so forth.

To Specify a Default Checkpoint Table in the GLOBALS File

This procedure specifies a global name for all checkpoint tables in the Oracle GoldenGate instance. You can override this name for any given Replicat group by specifying a different checkpoint table when you create the Replicat group.

1. Create a `GLOBALS` file (or edit the existing one, if applicable). The file name must be all capital letters on UNIX or Linux systems, without a file extension, and must reside in the root Oracle GoldenGate directory. You can use an ASCII text editor to create the file, making certain to observe the preceding naming conventions, or you can use GGSCI to create and save it with the correct name and location automatically. When using GGSCI, use the following command, typing `GLOBALS` in upper case.

```
EDIT PARAMS ./GLOBALS
```

2. Enter the following parameter:

```
CHECKPOINTTABLE container.owner.table
```

Where:

catalog.owner.table is the fully qualified name of the default checkpoint table, including the name of the container if the database is an Oracle multitenant container database (CDB).

3. Note the name of the table, then save and close the `GLOBALS` file. Make certain the file was created in the root Oracle GoldenGate directory. If there is a file extension, remove it.

Adjusting for Coordinated Replicat in Oracle RAC

If the Replicat for which you are creating a checkpoint table will run in an Oracle RAC configuration, it is recommended that you increase the `PCTFREE` attribute of the Replicat checkpoint table to as high a value as possible, as high as 90 if possible. This accommodates the more frequent checkpointing that is inherent in coordinated processing. This change must be made before starting the Replicat group for the first time. See [Creating an Online Replicat Group](#) for more information about coordinated Replicat.

Creating an Online Extract Group

To create an online Extract group, run GGSCI on the source system and issue the `ADD EXTRACT` command. Separate all command arguments with a comma. There are two syntax forms:

- [Syntax to Create a Regular, Passive, or Data Pump Extract Group](#)
- [Syntax to Create an Alias Extract Group](#)

Syntax to Create a Regular, Passive, or Data Pump Extract Group

```
ADD EXTRACT group
{, datasource}
{, BEGIN start_point} | {position_point}
[, PASSIVE]
[, THREADS n]
[, PARAMS pathname]
[, REPORT pathname]
[, DESC 'description']
```

Where:

- *group* is the name of the Extract group. A group name is required.
- *datasource* is required to specify the source of the data to be extracted. Use one of the following:
 - `TRANLOG` specifies the transaction log as the data source. When using this option for Oracle Enterprise Edition, you must issue the `DBLOGIN` command as the Extract database user (or a user with the same privileges) before using `ADD EXTRACT` (and also before issuing `DELETE EXTRACT` to remove an Extract group).

Use the *bsds* option for DB2 running on z/OS to specify the Bootstrap Data Set file name of the transaction log.
 - `INTEGRATED TRANLOG` specifies that this Extract will operate in integrated capture mode to receive logical change records (LCR) from an Oracle Database logmining server. This parameter applies only to Oracle Databases..
 - `EXTTRAILSOURCE trail name` to specify the relative or fully qualified name of a local trail. Use to create a data pump. A data pump can be used with any Oracle GoldenGate extraction method.
- `BEGIN start_point` defines an online Extract group by establishing an initial checkpoint and start point for processing. Transactions started before this point are discarded. Use one of the following:
 - `NOW` to begin extracting changes that are timestamped at the point when the `ADD EXTRACT` command is executed to create the group or, for an Oracle Extract in integrated mode, from the time the group is registered with the `REGISTER EXTRACT` command. Do not use `NOW` for a data pump Extract unless you want to bypass any data that was captured to the Oracle GoldenGate trail prior to the `ADD EXTRACT` statement.

YYYY-MM-DD HH:MM[:SS[.CCCCC]]) as the format for specifying an exact timestamp as the begin point. Use a begin point that is later than the time at which replication or logging was enabled.

- *position_point* specifies a specific position within a specific transaction log file at which to start processing. For the specific syntax to use for your database, see `ADD EXTRACT` in *Reference for Oracle GoldenGate*.
- `PASSIVE` indicates that the group is a passive Extract. When using `PASSIVE`, you must also use an alias Extract. This option can appear in any order among other `ADD EXTRACT` options.
- `THREADS n` is required only if Extract is operating in classic capture mode in an Oracle Real Application Cluster (RAC). It specifies the number of redo log threads being used by the cluster.
- `PARAMS pathname` is required if the parameter file for this group will be stored in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.
- `REPORT pathname` is required if the process report for this group will be stored in a location other than the `dirrpt` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.
- `DESC 'description'` specifies a description of the group.

Syntax to Create an Alias Extract Group

```
ADD EXTRACT group
, RMTHOST {host | IP address}
, {MGRPORT port} | {PORT port}
[, RMTNAME name]
[, DESC 'description']
```

Where:

- `RMTHOST` identifies this group as an alias Extract and specifies either the DNS name of the remote host or its IP address.
- `MGRPORT` specifies the port on the remote system where Manager is running. Use this option when using a dynamic Collector.
- `PORT` specifies a static Collector port. Use instead of `MGRPORT` only if running a static Collector.
- `RMTNAME` specifies the passive Extract name, if different from that of the alias Extract.
- `DESC 'description'` specifies a description of the group.

Example 13-1 Adding an Extract Group for Log-based Capture

This example creates an Extract group named `finance`. Extraction starts with records generated at the time when the group was created.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW
```

Example 13-2 Adding a Data-pump Extract Group

This example creates a data-pump Extract group named `finance`. It reads from the Oracle GoldenGate trail `c:\ggs\dirdat\lt`.

```
ADD EXTRACT finance, EXTRACTSOURCE c:\ggs\dirdat\lt
```

Example 13-3 Adding a Passive Extract Group

This example creates a passive Extract group named `finance`. Extraction starts with records generated at the time when the group was created. Because this group is marked as passive, an alias Extract on the target will initiate connections to this Extract.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW, PASSIVE
```

Example 13-4 Adding a Passive Data-pump Extract Group

This example creates a data-pump Extract group named `finance`. This is a passive data pump Extract that reads from the Oracle GoldenGate trail `c:\ggs\dir\dat\lt`. Because this data pump is marked as passive, an alias Extract on the target will initiate connections to it.

```
ADD EXTRACT finance, EXTTRAILSOURCE c:\ggs\dir\dat\lt, PASSIVE
```

Example 13-5 Adding an Alias Extract Group

This example creates an alias Extract group named `alias`.

```
ADD EXTRACT alias, RMTHOST sysA, MGRPORT 7800, RMTNAME finance
```

Example 13-6 Adding a Primary Extract in Integrated Mode for Oracle

This example creates an Extract in integrated capture mode for an Oracle source database and sets the start point to the time when the Extract group is registered with the Oracle database by means of the `REGISTER EXTRACT` command. Integrated capture is available only for an Oracle database.

```
ADD EXTRACT finance INTEGRATED TRANLOG, BEGIN NOW
```

Creating a Trail

After data has been extracted, it must be processed into one or more trails, where it is stored for processing by another Oracle GoldenGate process. A trail is a sequence of files that are created and aged as needed. Processes that read a trail are:

- Data-pump Extract: Extracts data from a local trail for further processing, if needed, and transfers it to the target system.
- Replicat: Reads a trail to apply change data to the target database.

You can create more than one trail to separate the data of different tables or applications, or to satisfy the requirements of a specific replication topology, such as a cascading topology. You link tables specified with a `TABLE` statement to a trail specified with an `EXTTRAIL` or `RMTTRAIL` parameter statement in the Extract parameter file. See [About the Oracle GoldenGate Trail](#) for detailed information about Oracle GoldenGate trails.

- [Assigning Storage for Oracle GoldenGate Trails](#)
- [Estimating Space for the Trails](#)
- [Adding a Trail](#)

Assigning Storage for Oracle GoldenGate Trails

In a typical configuration, there is at least one trail on the source system and one on the target system. Allocate enough disk space to allow for the following:

- The primary Extract process captures transactional data from the source database and writes it to the local trail. A data-pump Extract reads that trail and then transfers the data over the network to a remote trail on the target. If the network fails, the data pump fails but the primary Extract continues to process data to the local trail. There must be enough disk space to contain the data accumulation, or the primary Extract will abend.
- For a trail at the target location, provide enough disk space to handle data accumulation according to the purge rules set with the `PURGEOLDEXTRACTS` parameter. Even with `PURGEOLDEXTRACTS` in use, data will always accumulate on the target because it is transferred across the network faster than it can be applied to the target database.

To prevent trail activity from interfering with business applications, assign a separate disk or file system to contain the trail files. Trail files can reside on drives that are local to the Oracle GoldenGate installation, or they can reside on NAS or SAN devices. In an Oracle cluster, they can reside on ASM or DBFS storage.

Estimating Space for the Trails

The following are guidelines for estimating the amount of disk space that will be required to store Oracle GoldenGate trail data.

1. Estimate the longest time that the network could be unavailable. Plan to store enough data to withstand the longest possible outage, because otherwise you will need to resynchronize the source and target data if the outage outlasts disk capacity.
2. Estimate how much transaction log volume your business applications generate in one hour.
3. Use the following formula to calculate the required disk space.

[log volume in one hour] x [number of hours downtime] x .4 = trail disk space

This equation uses a multiplier of 40 percent because only about 40 percent of the data in a transaction log is needed by Oracle GoldenGate.

Note:

This formula is a conservative estimate, and you should run tests once you have configured Oracle GoldenGate to determine exactly how much space you need.

Adding a Trail

When you create, or *add*, a trail, you do not physically create any files on disk. The files are created automatically by an Extract process. Rather, you specify the name of the trail and associate it with the Extract group that writes to it.

To add a trail, issue the following command in GGSCI on the source system.

```
ADD {RMTTRAIL | EXTTRAIL} pathname, EXTRACT group  
[, MEGABYTES n]
```

Where:

- `RMTTRAIL` specifies a trail on a remote system.
- `EXTTRAIL` specifies a trail on the local system.
 - `EXTTRAIL` cannot be used for an Extract in `PASSIVE` mode.
 - `EXTTRAIL` must be used to specify a local trail that is read by a data pump.
- *pathname* is the relative or fully qualified name of the trail, including a two-character name that can be any two alphanumeric characters, for example `c:\ggs\dir\rt`. Oracle GoldenGate appends a serial number to each trail file as it is created during processing. Typically, trails are stored in the `dir\dat` sub-directory of the Oracle GoldenGate directory.
- `EXTRACT group` specifies the name of the Extract group that writes to this trail. Only one Extract group can write to a trail.
- `MEGABYTES n` is an optional argument with which you can set the size, in megabytes, of each trail file (default is 100).

Example 13-7 Creating a Local Trail

This example creates a local trail named `/ggs/dir\dat\lt` for Extract group `ext`.

```
ADD EXTTRAIL /ggs/dir\dat\lt, EXTRACT ext
```

Example 13-8 Creating a Remote Trail

This example creates a trail named `c:\ggs\dir\dat\rt` for Extract group `finance`, with each file sized at approximately 50 megabytes.

```
ADD RMTTRAIL c:\ggs\dir\dat\rt, EXTRACT finance, MEGABYTES 200
```

Creating a Parameter File for Online Extraction

Follow these instructions to create a parameter file for an online Extract group. A parameter file is not required for an alias Extract group.

1. In GGSCI on the source system, issue the following command.

```
EDIT PARAMS name
```

Where:

name is either the name of the Extract group that you created with the `ADD EXTRACT` command or the fully qualified name of the parameter file if you defined an alternate location when you created the group.

2. Enter the parameters in [Creating a Parameter File for Online Extraction](#) in the order shown, starting a new line for each parameter statement. Some parameters apply only for certain configurations.

Table 13-1 Online Change-Extraction Parameters

Parameter	Description
EXTRACT <i>group</i>	Configures Extract as an online process with checkpoints.
<ul style="list-style-type: none"> <i>group</i> is the name of the Extract group that you created with the ADD EXTRACT command. 	
[SOURCEDB <i>dsn</i> <i>container</i> <i>catalog</i>] [, USERIDALIAS <i>alias options</i> , USERID <i>user, options</i>]	<p>Specifies database connection information.</p> <p>SOURCEDB specifies the source data source name (DSN). See for more information.</p> <p>USERID and USERIDALIAS specify database credentials if required.</p> <p>The database connection can be omitted if the group is a data pump on an intermediary system that does not have a database. In this case, there can be no column mapping or conversion performed.</p>
RMTHOSTOPTIONS <i>host</i> , MGRPORT <i>port</i> , [, ENCRYPT <i>algorithm</i> KEYNAME <i>key_name</i>]	<p>Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP. Only required when sending data over IP to a remote system (if ADD RMTTRAIL was used to create the trail). Not required if the trail is on the local system (if ADD EXTTRAIL was used).</p> <p>Not valid for a passive Extract group.</p>
ENCRYPTTRAIL <i>algorithm</i>	Encrypts all trails that are specified after this entry.
DECRYPTTRAIL	(For a data pump) Decrypts the data in the input trail. Use only if the data pump must process the data before writing it to the output trail.
RMTTRAIL <i>pathname</i> EXTTRAIL <i>pathname</i>	<p>Specifies a trail. If specifying multiple trails, follow each designation with the appropriate TABLE statements.</p> <p>EXTTRAIL is not valid for a passive Extract group.</p> <p>If trails or files will be of different versions, use the FORMAT option of RMTTRAIL or EXTTRAIL. See EXTTRAIL in <i>Reference for Oracle GoldenGate</i></p>
<ul style="list-style-type: none"> Use RMTTRAIL to specify the relative or fully qualified name of a remote trail created with the ADD RMTTRAIL command. Use EXTTRAIL to specify the relative or fully qualified name of a local trail created with the ADD EXTTRAIL command (to be read by a data pump or VAM-sort Extract). 	
LOGALLSUPCOLS	Use when using integrated Replicat for an Oracle target, or when using Conflict Detection and Resolution (CDR) support. Writes the before images of scheduling columns to the trail. (Scheduling columns are primary key, unique index, and foreign key columns.) See LOGALLSUPCOLS in <i>Reference for Oracle GoldenGate</i> .

Table 13-1 (Cont.) Online Change-Extraction Parameters

Parameter	Description
SOURCECATALOG	Specifies a default container in an Oracle multitenant container database or SEQUENCE statements. Enables the use of two-part names (<i>schema.object</i>) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of TABLE or SEQUENCE parameters.
SEQUENCE <i>[container.]owner.sequence;</i>	Specifies the fully qualified name of an Oracle sequence to capture. Include the container name if the database is a multitenant container database (CDB).
TABLE <i>[container. catalog.]owner.object;</i>	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container or catalog unless SOURCECATALOG is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
CATALOGEXCLUDE SCHEMAEXCLUDE TABLEEXCLUDE EXCLUDEWILDCARDOBJECTSONLY	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated TABLE statement.

- Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate Parameters in *Reference for Oracle GoldenGate*.
- Save and close the parameter file.

Parameter	Description
VAM <i>library</i> , PARAMS (' <i>param</i> ' [, ' <i>param</i> '] [, ...])	Valid only for an Extract group that interfaces with a Teradata Access Module. Supplies the name of the library and parameters that must be passed to the Oracle GoldenGate API, such as the name of the TAM initialization file and the program that interacts with the library as the callback library. Example: VAM vam.dll, PARAMS ('inifile', 'vamerge1.ini', 'callbacklib', 'extract.exe')
	NA

Creating an Online Replicat Group

Before creating a Replicat group, you should evaluate which of the Replicat modes is appropriate for your environment: *classic mode* (also known as *nonintegrated mode* in Oracle environments), *coordinated mode*, and *integrated mode*.

Topics:

- [About Classic Replicat Mode](#)
- [About Coordinated Replicat Mode](#)

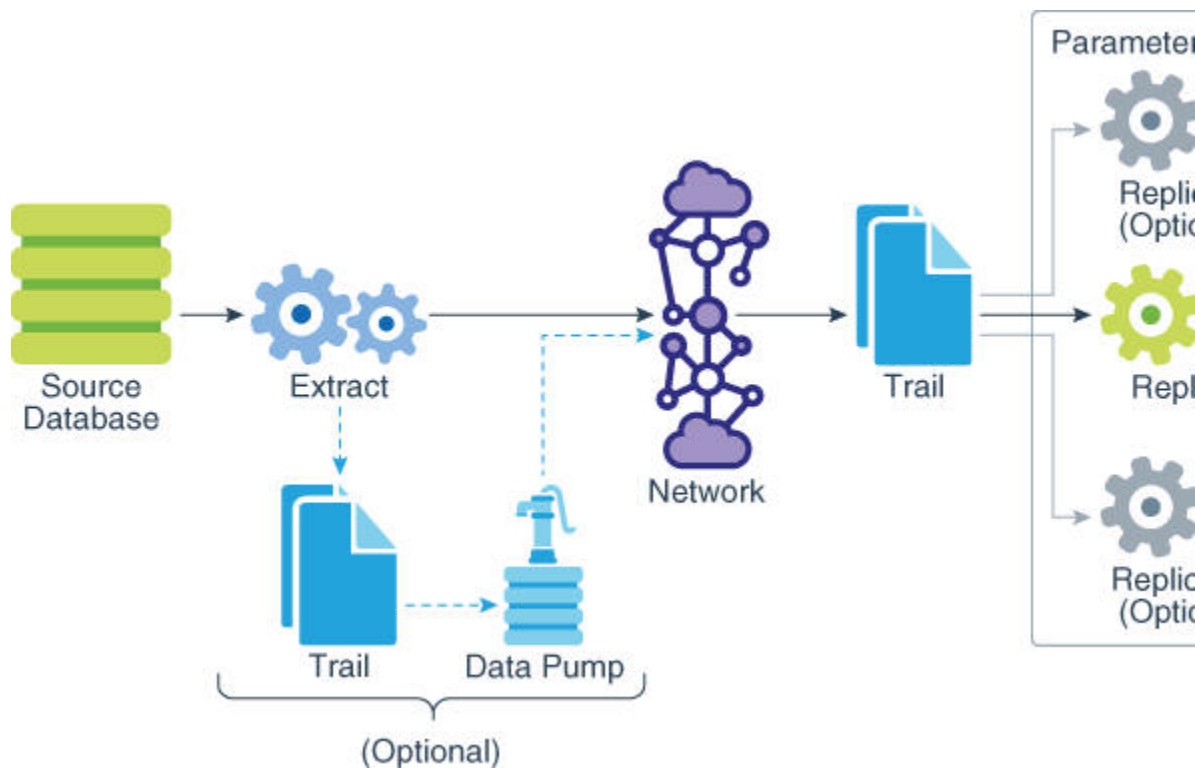
- [About Integrated Replicat Mode](#)
- [Understanding Replicat Processing in Relation to Parameter Changes](#)
- [Creating the Replicat Group](#)

About Classic Replicat Mode

In classic mode, Replicat is a single-threaded process that uses standard SQL to apply data to the target tables. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Constructs SQL statements that represent source database DML or DDL transactions (in committed order).
- Applies the SQL to the target through the SQL interface that is supported for the given target database, such as ODBC or the native database interface.

Figure 13-1 Classic Replicat



As shown in [Figure 13-1](#), you can apply transactions in parallel with a classic Replicat, but only by partitioning the workload across multiple Replicat processes. A parameter file must be created for each Replicat.

To determine whether to use classic mode for any objects, you must determine whether the objects in one Replicat group will ever have dependencies on objects in any other Replicat group, transactional or otherwise. Not all workloads can be partitioned across multiple Replicat groups and still preserve the original transaction atomicity. For example, tables for which the workload routinely updates the primary

key cannot easily be partitioned in this manner. DDL replication (if supported for the database) is not viable in this mode, nor is the use of some `SQLEXEC` or `EVENTACTIONS` features that base their actions on a specific record.

If your tables do not have any foreign- key dependencies or updates to primary keys, classic mode may be suitable. Classic mode requires less overhead than coordinated mode.

For more information about using parallel Replicat groups, see [Tuning the Performance of Oracle GoldenGate](#).

About Coordinated Replicat Mode

In coordinated mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Processes operations sent to each thread in a committed order.
- Applies the SQL to the target through the SQL interface that is supported for the given target database, such as ODBC or the native database interface.

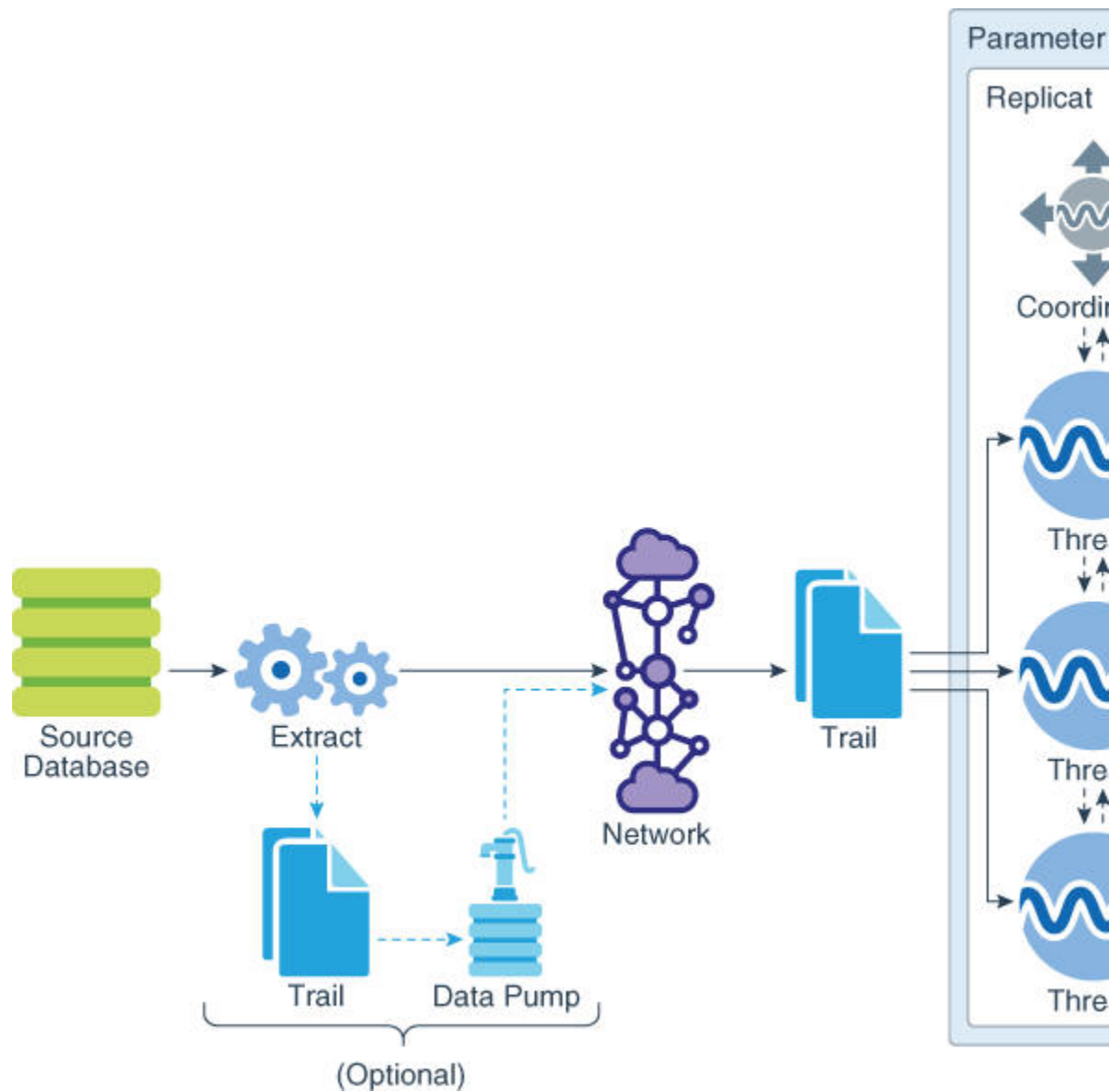
The difference between classic mode and coordinated mode is that Replicat is multi-threaded in coordinated mode. Within a single Replicat instance, multiple threads read the trail independently and apply transactions in parallel. Each thread handles the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A *coordinator* thread coordinates the transactions across threads to account for dependencies among the threads.

The source transactions could be split across CR processes such that the integrity of the total source transaction is not maintained. The portion of the transaction processed by a CR process is done in committed order but the whole transaction across all CR processes is not.

Coordinated Replicat allows for user-defined partitioning of the workload so as to apply high volume transactions concurrently. In addition, it automatically coordinates the execution of transactions that require coordination, such as DDL, and primary key updates with `THREADRANGE` partitioning. Such a transaction is executed as one transaction in the target with full synchronization: it waits until all prior transactions are applied first, and all transactions after this barrier transaction have to wait until this barrier transaction is applied.

Only one parameter file is required for a coordinated Replicat, regardless of the number of threads. You use the `THREAD` or `THREADRANGE` option in the `MAP` statement to specify which threads process the transactions for those objects, and you specify the maximum number of threads when you create the Replicat group.

Figure 13-2 Coordinated Replicat



- [About Barrier Transactions](#)
- [How Barrier Transactions are Processed](#)
- [About the Global Watermark](#)

About Barrier Transactions

Barrier transactions are managed automatically in a coordinated Replicat configuration. Barrier transactions are transactions that require coordination across threads. Examples include DDL statements, transactions that include updates to primary keys, and certain `EVENTACTIONS` actions.

Optionally, you can force other transactions to be treated like a barrier transaction through the use of the `COORDINATED` keyword in a `MAP` statement. One use case for this

would be force a `SQLEXEC` to be executed in a manner similar to a serial execution. This could be beneficial if the results can become ambiguous unless the state of the target is consistent across all transactions.

 **Note:**

Coordinated Replicat doesn't do dependency calculations for non-barrier transactions when a mapped table is partitioned based on `THNREADRANGE`. It relies on specified `THREADRANGE` columns to compute a hash value. It partitions the incoming data based on the hash value and sends all the records that match this hash value to same thread.

How Barrier Transactions are Processed

All threads converge and wait at the start of a barrier transaction. The barrier transaction is suspended until the other threads reach its start position. If any threads were already processing part of the barrier transaction, those threads perform a rollback. Grouped transactions, such as those controlled by the `BATCHSQL` or `GROUPTRANSOPS` parameters, are also rolled back and then reapplied until they reach the start of the barrier transaction.

All of the threads converge and wait at the start of the next transaction after the barrier transaction as well. The two synchronization points, before and after the barrier transaction, ensure that metadata operations and `EVENTACTIONS` actions all occur in the proper order relevant to the data operations.

Once the threads are synchronized at the start of the barrier transaction, the barrier transaction is processed serially by the thread that has the lowest thread ID among all of the threads specified in the `MAP` statements, and then parallel processing across threads is resumed. You can force barrier transactions to be processed through a specific thread, which is always thread 0, by specifying the `USEDEDICATEDCOORDINATIONTHREAD` parameter in the Replicat parameter file.

About the Global Watermark

A clean shutdown of a coordinated Replicat ensures that all threads stop at the same transaction boundary in the trail, known as the *global watermark*. This is defined as the synchronized point where all records before this position were either committed or ignored by all of their respective threads. If a clean shutdown is not possible, you can use the `SYNCHRONIZE REPLICAT` command to return all of the threads to the position of the thread that made the most recent checkpoint. See [Synchronizing Threads After an Unclean Stop](#) for more information about recovering a coordinated Replicat group.

 **Note:**

Coordinated Replicat is an online process only. Do not use it to perform initial loads.

About Integrated Replicat Mode

In integrated mode, available for Oracle databases of version 11.2.0.4 or later, Replicat leverages the apply processing functionality that is available within the target Oracle database. In this mode, Replicat reads the trail, constructs logical change records that represent source DML or DDL transactions, and transmits these records to an inbound server in the Oracle target database. The inbound server applies the data to the target database.

For more information about using integrated Replicat, see *About Integrated Mode in Using Oracle GoldenGate for Oracle Database*.

 **Note:**

Integrated Replicat is an online process only. Do not use it to perform initial loads.

Understanding Replicat Processing in Relation to Parameter Changes

Changes to the object specifications in the Replicat configuration cannot be made to affect transactions that are already applied, but only for those not yet applied. This is an important consideration when using coordinated or integrated Replicat.

For a Replicat in classic mode, the boundary between applied and non-applied transactions is a clean one, because transactions are applied serially. For a coordinated or integrated Replicat, however, there is no single point in the trail that marks applied and unapplied transactions, because transactions are being applied asynchronously in parallel.

In coordinated or integrated modes, there are a low watermark, below which all transactions were applied, and a high watermark above which no transactions were applied. In between those boundaries there may be transactions that may or may not have been applied, depending on the progress of individual threads. As a result, if Replicat is forced changes to object specifications in the Replicat configuration may be reflected unevenly in the target after Replicat is restarted. Examples of parameter changes for which this applies are changes to `MAP` mappings, `FILTER` clauses, and `EXCLUDE` parameters.

Changes to the Replicat configuration should not be made after Replicat abends or is forcibly terminated. Replicat should be allowed to recover to its last checkpoint after startup. For coordinated Replicat, you can follow the administrative procedures in [Administering a Coordinated Replicat Configuration](#). Once the recovery is complete, Replicat can be shut down gracefully with the `STOP REPLICAT` command, and then you can make the changes to the object specifications.

Creating the Replicat Group

To create an online Replicat group, run GGSCI on the target system and issue the `ADD REPLICAT` command. Separate all command arguments with a comma.

```
ADD REPLICAT group, EXTTRAIL path  
[, {INTEGRATED | COORDINATED [MAXTHREADS number]}]
```

```
[, BEGIN start_point | , EXTSEQNO seqno, EXTRBA rba]
[, CHECKPOINTTABLE owner.table]
[, NOBCKEPOINT]
[, PARAMS path]
[, REPORT path]
```

Where:

- *group* is the name of the Replicat group. A group name is required. See [Naming Conventions for Processes](#) for Oracle GoldenGate naming conventions.
- *EXTTRAIL path* is the relative or fully qualified name of the trail that you defined with the `ADD RMTTRAIL` command.
- `INTEGRATED` specified that this Replicat group will operate in integrated mode. This mode is available for Oracle databases..
- `COORDINATED` specifies that this Replicat group will operate in coordinated mode. `MAXTHREADS` specifies the maximum number of threads allowed for this group. Valid values are from 1 through 500. `MAXTHREADS` is optional. The default number of threads without `MAXTHREADS` is 25.

 **Note:**

Each Replicat thread is considered a Replicat group in the context of the `MAXGROUPS` parameter. `MAXGROUPS` controls the maximum number of process groups allowed in the Oracle GoldenGate instance. `MAXTHREADS` plus the number of other process groups in the Oracle GoldenGate instance must not exceed the value set with `MAXGROUPS` (default is 1000).

- `BEGIN start_point` defines an online Replicat group by establishing an initial checkpoint and start point for processing. Use one of the following:
 - `NOW` to begin replicating changes timestamped at the point when the `ADD REPLICAT` command is executed to create the group.
 - `YYYY-MM-DD HH:MM[:SS[.CCCCC]]` as the format for specifying an exact timestamp as the begin point.
- `EXTSEQNO seqno, EXTRBA rba` specifies the sequence number of the file in a trail in which to begin reading data and the relative byte address within that file. By default, processing begins at the beginning of a trail unless this option is used. For the sequence number, specify the number, but not any zeroes used for padding. For example, if the trail file is `c:\ggs\dirdat\aa000000026`, specify `EXTSEQNO 26`. Contact Oracle Support before using this option.
- `CHECKPOINTTABLE owner.table` specifies the owner and name of a checkpoint table other than the default specified in the `GLOBALS` file. To use this argument, you must add the checkpoint table to the database with the `ADD CHECKPOINTTABLE` command (see [Creating a Checkpoint Table](#)).
- `NOBCKEPOINT` specifies that this Replicat group will not use a checkpoint table.
- `PARAMS path` is required if the parameter file for this group will be stored in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.

- `REPORT path` is required if the process report for this group will be stored in a location other than the `dirrpt` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.

Example 13-9 Creating an Online Replicat Group

This example creates an online Replicat group named `finance` and specifies a trail of `c:\ggs\dirdat\rt`. The parameter file is stored in the alternate location of `\ggs\params`, and the report file is stored in its default location.

```
ADD REPLICAT finance, EXTTRAIL c:\ggs\dirdat\rt, PARAMS \ggs\params
```

Creating a Parameter File for Online Replication

Follow these instructions to create a parameter file for an online Replicat group.

1. In GGSCI on the target system, issue the following command.

```
EDIT PARAMS name
```

Where:

`name` is either the name of the Replicat group that you created with the `ADD REPLICAT` command or the fully qualified name of the parameter file if you defined an alternate location when you created the group.

2. Enter the parameters listed in [Table 13-2](#) in the order shown, starting a new line for each parameter statement.

Table 13-2 Online Change-Replication Parameters

Parameter	Description
<code>REPLICAT group</code>	Configures Replicat as an online process with checkpoints.
<ul style="list-style-type: none"> • <code>group</code> is the name of the Replicat group that you created with the <code>ADD REPLICAT</code> command. 	
<code>{SOURCEDEFS path} ASSUMETARGETDEFS</code>	Specifies how to interpret data definitions. For Oracle databases that use multi-byte character sets, you must use <code>SOURCEDEFS</code> (with a <code>DEFGEN</code> -generated definitions file) if the source semantics setting is in bytes and the target is in characters. This is required even when the source and target data definitions are identical. See Associating Replicated Data with Metadata , for more information.
<ul style="list-style-type: none"> • Use <code>SOURCEDEFS</code> if the source and target tables have different definitions. Specify the source data-definitions file generated by <code>DEFGEN</code>. See Associating Replicated Data with Metadata, for more information. • Use <code>ASSUMETARGETDEFS</code> if the source and target tables have the same definitions. 	

Table 13-2 (Cont.) Online Change-Replication Parameters

Parameter	Description
<pre>[DEFERAPPLYINTERVAL <i>n unit</i>]</pre> <ul style="list-style-type: none"> <i>n</i> is a numeric value for the amount of time to delay before applying transactions. Minimum is set by the EOFDELAY parameter. Maximum is seven days. <i>unit</i> can be: S SEC SECS SECOND SECONDS MIN MINS MINUTE MINUTES HOUR HOURS DAY DAYS 	<p>Optional. Specifies an amount of time for Replicat to wait before applying its transactions to the target system.</p>
<pre>[TARGETDB <i>dsn container catalog</i>] [, USERIDALIAS <i>alias options</i> , USERID <i>user, options</i>]</pre>	<p>Specifies database connection information.</p> <p>TARGETDB specifies the target datasource name (DSN). See TARGETDB in <i>Reference for Oracle GoldenGate</i> for more information .</p> <p>USERID and USERIDALIAS specify database credentials if required.</p>
HANDLECOLLISIONS	<p>Specifies collision handling. Use only if you are performing an initial load concurrently with starting online processing and the source database will remain active during the load. HANDLECOLLISIONS resolves the results of the copy with the ongoing replicated transactional changes. It resolves insert operations for which the row already exists and update and delete operations for which the row does not exist. It can be used globally for all MAP statements in a parameter file or within a MAP statement, or both.</p>
SOURCECATALOG	<p>Specifies a default container in a source Oracle multitenant container database. Enables the use of two-part names (<i>schema.object</i>) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of MAP parameters.</p>

Table 13-2 (Cont.) Online Change-Replication Parameters

Parameter	Description
<pre>MAP [container. catalog.]owner.object, TARGET owner.object[, DEF template] [THREAD (thread_ID)] [THREADRANGE (thread_range[, column_list])] [COORDINATED] ;</pre>	<p>Specifies a relationship between a source object or objects and a target object or objects. MAP specifies the source object, and TARGET specifies the target object.</p> <p>For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database the source object name must include the name of the container or catalog unless SOURCECATALOG is used.</p> <p>For the target object, specify only the <i>owner.object</i> components of the name, regardless of the type of database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to apply data. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.</p> <p>The THREAD, THREADRANGE, and COORDINATED options are valid for Replicat when in coordinated mode. They enable you to partition the workload to one or more specific Replicat threads. See in <i>Reference for Oracle GoldenGate</i> for syntax and usage.</p> <p>The DEF option specifies a definitions template. See Associating Replicated Data with Metadata for more information about data definitions.</p>
<pre>CATALOGEXCLUDE SCHEMAEXCLUDE MAPEXCLUDE EXCLUDEWILDCARDOBJECTSONLY</pre>	<p>Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated MAP statement.</p>

1. Enter any appropriate optional Replicat parameters listed in Summary of Replicat Parameters in *Reference for Oracle GoldenGate*.
2. Save and close the file.

 **Note:**

If using integrated Replicat for Oracle, see [Understanding Replicat Processing in Relation to Parameter Changes](#) for important information about making configuration changes to Replicat once processing is started.

14

Handling Processing Errors

This chapter describes how to configure the Oracle GoldenGate processes to handle errors.

Oracle GoldenGate reports processing errors in several ways by means of its monitoring and reporting tools. For more information about these tools, see [Monitoring Oracle GoldenGate Processing](#).

Topics:

- [Overview of Oracle GoldenGate Error Handling](#)
- [Handling Extract Errors](#)
- [Handling Replicat Errors during DML Operations](#)
- [Handling Replicat errors during DDL Operations](#)
- [Handling TCP/IP Errors](#)
- [Maintaining Updated Error Messages](#)
- [Resolving Oracle GoldenGate Errors](#)

Overview of Oracle GoldenGate Error Handling

Oracle GoldenGate provides error-handling options for:

- Extract
- Replicat
- TCP/IP

Handling Extract Errors

There is no specific parameter to handle Extract errors when DML operations are being extracted, but Extract does provide a number of parameters that can be used to prevent anticipated problems. These parameters handle anomalies that can occur during the processing of DML operations, such as what to do when a row to be fetched cannot be located, or what to do when the transaction log is not available. The following is a partial list of these parameters.

- FETCHOPTIONS
- WARNLONGTRANS
- DBOPTIONS
- TRANLOGOPTIONS

To handle extraction errors that relate to DDL operations, use the `DDLERROR` parameter.

For a complete parameter list, see *Reference for Oracle GoldenGate*.

Handling Replicat Errors during DML Operations

To control the way that Replicat responds to an error during one of its DML statements, use the `REPERROR` parameter in the Replicat parameter file. You can use `REPERROR` as a global parameter or as part of a `MAP` statement. You can handle most errors in a default fashion (for example, to cease processing) with `DEFAULT` and `DEFAULT2` options, and also handle other errors in a specific manner.

The following comprise the range of `REPERROR` responses:

- `ABEND`: roll back the transaction and stop processing.
- `DISCARD`: log the error to the discard file and continue processing.
- `EXCEPTION`: send the error for exceptions processing. See [Handling Errors as Exceptions](#) for more information.
- `IGNORE`: ignore the error and continue processing.
- `RETRYOP [MAXRETRIES n]`: retry the operation, optionally up to a specific number of times.
- `TRANSABORT [, MAXRETRIES n] [, DELAY[C]SECS n]`: abort the transaction and reposition to the beginning, optionally up to a specific number of times at specific intervals.
- `RESET`: remove all previous `REPERROR` rules and restore the default of `ABEND`.
- `TRANSDISCARD`: discard the entire replicated source transaction if any operation within that transaction, including the commit, causes a Replicat error that is listed in the error specification. This option is useful when integrity constraint checking is disabled on the target.
- `TRANSEXCEPTION`: perform exceptions mapping for every record in the replicated source transaction, according to its exceptions-mapping statement, if any operation within that transaction (including the commit) causes a Replicat error that is listed in the error specification.

Most options operate on the individual record that generated an error, and Replicat processes the other, successful operations in the transaction. The exceptions are `TRANSDISCARD` and `TRANSEXCEPTION`: These options affect all records in a transaction if any record in that transaction generates an error. (The `ABEND` option also applies to the entire transaction, but does not apply error handling.)

See *Reference for Oracle GoldenGate* for `REPERROR` syntax and usage.

- [Handling Errors as Exceptions](#)

Handling Errors as Exceptions

When the action of `REPERROR` is `EXCEPTION` or `TRANSEXCEPTION`, you can map the values of operations that generate errors to an exceptions table and, optionally, map other information about the error that can be used to resolve the error. See [About the Exceptions Table](#).

To map the exceptions to the exceptions table, use either of the following options of the `MAP` parameter:

- `MAP` with `EXCEPTIONSONLY`

- MAP with MAPEXCEPTION
- Using EXCEPTIONSONLY
- Using MAPEXCEPTION
- About the Exceptions Table

Using EXCEPTIONSONLY

EXCEPTIONSONLY is valid for one pair of source and target tables that are explicitly named and mapped one-to-one in a MAP statement; that is, there cannot be wildcards. To use EXCEPTIONSONLY, create two MAP statements for each source table that you want to use EXCEPTIONSONLY for on the target:

- The first, a standard MAP statement, maps the source table to the actual target table.
- The second, an *exceptions MAP statement*, maps the source table to the *exceptions table* (instead of to the target table). An exceptions MAP statement executes immediately after an error on the source table to send the row values to the exceptions table.

To identify a MAP statement as an exceptions MAP statement, use the INSERTALLRECORDS and EXCEPTIONSONLY options. The exceptions MAP statement must immediately follow the regular MAP statement that contains the same source table. Use a COLMAP clause in the exceptions MAP statement if the source and exceptions-table columns are not identical, or if you want to map additional information to extra columns in the exceptions table, such as information that is captured by means of column-conversion functions or SQLEXEC.

For more information about these parameters, see *Reference for Oracle GoldenGate*.

- A regular MAP statement that maps the source table `ggs.equip_account` to its target table `equip_account2`.
- An exceptions MAP statement that maps the same source table to the exceptions table `ggs.equip_account_exception`.

In this case, four extra columns were created, in addition to the same columns that the table itself contains:

```
DML_DATE
OPTYPE
DBERRNUM
DBERRMSG
```

To populate the DML_DATE column, the @DATENOW column-conversion function is used to get the date and time of the failed operation, and the result is mapped to the column. To populate the other extra columns, the @GETENV function is used to return the operation type, database error number, and database error message.

The EXCEPTIONSONLY option of the exceptions MAP statement causes the statement to execute only after a failed operation on the source table. It prevents every operation from being logged to the exceptions table.

The INSERTALLRECORDS parameter causes all failed operations for the specified source table, no matter what the operation type, to be logged to the exceptions table as *inserts*.

**Note:**

There can be no primary key or unique index restrictions on the exception table. Uniqueness violations are possible in this scenario and would generate errors.

Example 14-1 EXCEPTIONSONLY

This example shows how to use `REPERROR` with `EXCEPTIONSONLY` and an exceptions `MAP` statement. This example only shows the parameters that relate to `REPERROR`; other parameters not related to error handling are also required for Replicat.

```
REPERROR (DEFAULT, EXCEPTION)
MAP ggs.equip_account, TARGET ggs.equip_account2,
COLMAP (USEDEFAULTS);
MAP ggs.equip_account, TARGET ggs.equip_account_exception,
EXCEPTIONSONLY,
INSERTALLRECORDS
COLMAP (USEDEFAULTS,
DML_DATE = @DATENOW (),
OPTYPE = @GETENV ('LASTERR', 'OPTYPE'),
DBERRNUM = @GETENV ('LASTERR', 'DBERRNUM'),
DBERRMSG = @GETENV ('LASTERR', 'DBERRMSG'));
```

In this example, the `REPERROR` parameter is set for `DEFAULT` error handling, and the `EXCEPTION` option causes the Replicat process to treat failed operations as exceptions and continue processing.

Using MAPEXCEPTION

`MAPEXCEPTION` is valid when the names of the source and target tables in the `MAP` statement are wildcarded. Place the `MAPEXCEPTION` clause in the regular `MAP` statement, the same one where you map the source tables to the target tables. Replicat maps all operations that generate errors from all of the wildcarded tables to the same exceptions table; therefore, the exceptions table should contain a superset of all of the columns in all of the wildcarded tables.

Because you cannot individually map columns in a wildcard configuration, use the `COLMAP` clause with the `USEDEFAULTS` option to handle the column mapping for the wildcarded tables (or use the `COLMATCH` parameter if appropriate), and use explicit column mappings to map any additional information, such as that captured with column-conversion functions or `SQLEXEC`.

When using `MAPEXCEPTION`, include the `INSERTALLRECORDS` parameter in the `MAPEXCEPTION` clause. `INSERTALLRECORDS` causes all operation types to be applied to the exceptions table as `INSERT` operations. This is required to keep an accurate record of the exceptions and to prevent integrity errors on the exceptions table.

For more information about these parameters, see *Reference for Oracle GoldenGate*.

Example 14-2 MAPEXCEPTION

This is an example of how to use `MAPEXCEPTION` for exceptions mapping. The `MAP` and `TARGET` clauses contain wildcarded source and target table names. Exceptions that

occur when processing any table with a name beginning with TRX are captured to the `fin.trxexceptions` table using the designated mapping.

```
MAP src.trx*, TARGET trg.*,
MAPEXCEPTION (TARGET fin.trxexceptions,
INSERTALLRECORDS,
COLMAP (USEDEFAULTS,
ACCT_NO = ACCT_NO,
OPTYPE = @GETENV ('LASTERR', 'OPTYPE'),
DBERR = @GETENV ('LASTERR', 'DBERRNUM'),
DBERRMSG = @GETENV ('LASTERR', 'DBERRMSG')
)
);
```

About the Exceptions Table

Use an exceptions table to capture information about an error that can be used for such purposes as troubleshooting your applications or configuring them to handle the error. At minimum, an exceptions table should contain enough columns to receive the entire row image from the failed operation. You can define extra columns to contain other information that is captured by means of column-conversion functions, `SQLEXEC`, or other external means.

To ensure that the trail record contains values for all of the columns that you map to the exceptions table, you can use either the `LOGALLSUPCOLS` parameter or the following parameters in the Extract parameter file:

- Use the `NOCOMPRESSDELETES` parameter so that all columns of a row are written to the trail for `DELETE` operations.
- Use the `GETUPDATEBEFORES` parameter so that Extract captures the before image of a row and writes them to the trail.

For more information about these parameters, see *Reference for Oracle GoldenGate*.

Handling Replicat errors during DDL Operations

To control the way that Replicat responds to an error that occurs for a DDL operation on the target, use the `DDLERROR` parameter in the Replicat parameter file. For more information, see *Reference for Oracle GoldenGate*.

Handling TCP/IP Errors

To provide instructions for responding to TCP/IP errors, use the `TCPERRS` file. This file is in the Oracle GoldenGate directory

Table 14-1 TCPERRS Columns

Column	Description
Error	Specifies a TCP/IP error for which you are defining a response.
Response	Controls whether or not Oracle GoldenGate tries to connect again after the defined error. Valid values are either <code>RETRY</code> or <code>ABEND</code> .

Table 14-1 (Cont.) TCPERRS Columns

Column	Description
Delay	Controls how long Oracle GoldenGate waits before attempting to connect again.
Max Retries	Controls the number of times that Oracle GoldenGate attempts to connect again before aborting.

If a response is not explicitly defined in the TCPERRS file, Oracle GoldenGate responds to TCP/IP errors by abending.

Example 14-3 TCPERRS File

```
# TCP/IP error handling parameters
# Default error response is abend
#
# Error          Response   Delay(csecs)  Max Retries
ECONNABORTED    RETRY       1000          10
ECONNREFUSED    RETRY       1000          12
ECONNRESET      RETRY       500           10
ENETDOWN        RETRY       3000          50
ENETRESET       RETRY       1000          10
ENOBUFS         RETRY       100           60
ENOTCONN        RETRY       100           10
EPIPE           RETRY       500           10
ESHUTDOWN       RETRY       1000          10
ETIMEDOUT       RETRY       1000          10
NODYNPORTS     RETRY       100           10
```

The TCPERRS file contains default responses to basic errors. To alter the instructions or add instructions for new errors, open the file in a text editor and change any of the values in the columns shown in [Table 14-1](#):

Maintaining Updated Error Messages

The error, information, and warning messages that Oracle GoldenGate processes generate are stored in a data file named `ggmessage.dat` in the Oracle GoldenGate installation directory. The version of this file is checked upon process startup and must be identical to that of the process in order for the process to operate.

Resolving Oracle GoldenGate Errors

To get help with specific troubleshooting issues, go to My Oracle Support at <http://support.oracle.com> and search the Knowledge Base.

Instantiating Oracle GoldenGate with an Initial Load

This chapter describes running an initial data load to instantiate the replication environment.

The initial load can be done in Classic Architecture and in Microservices Architecture.

- [Overview of the Initial-Load Procedure](#)
- [Initial Load in Classic Architecture](#)
In Classic Architecture you can load data using various options. The processes and steps do so, are described in this topic.

Overview of the Initial-Load Procedure

You can use Oracle GoldenGate to:

- Perform a standalone batch load to populate database tables for migration or other purposes.
- Load data into database tables as part of an initial synchronization run in preparation for change synchronization with Oracle GoldenGate.
- [Improving the Performance of an Initial Load](#)
- [Prerequisites for Initial Load](#)

Improving the Performance of an Initial Load

For all initial load methods except those performed with a database utility, you can load large databases more quickly by using parallel Oracle GoldenGate processes. To use parallel processing, take the following steps.

1. Follow the directions in this chapter for creating an initial-load Extract and an initial-load Replicat for each set of parallel processes that you want to use.
2. With the `TABLE` and `MAP` parameters, specify a different set of tables for each pair of Extract-Replicat processes, or you can use the `SQLPREDICATE` option of `TABLE` to partition the rows of large tables among the different Extract processes.

For all initial load methods, testing has shown that using the `TCPBUFSIZE` option in the `RMTHOST` parameter produced three times faster throughput than loads performed without it. Do not use this parameter if the target system is NonStop.

Prerequisites for Initial Load

Verify that you meet the prerequisites for executing an initial load that are described in the following sections.

- [Disable DDL Processing](#)

- [Prepare the Target Tables](#)
- [Configure the Manager Process](#)
- [Create a Data-definitions File](#)
- [Create Change-synchronization Groups](#)
- [Sharing Parameters between Process Groups](#)

Disable DDL Processing

Before executing an initial load, disable DDL extraction and replication. DDL processing is controlled by the `DDL` parameter in the Extract and Replicat parameter files.

Prepare the Target Tables

The following are suggestions that can make the load go faster and help you to avoid errors.

- **Data:** Make certain that the target tables are empty. Otherwise, there may be duplicate-row errors or conflicts between existing rows and rows that are being loaded.
- **Constraints:** Disable foreign-key constraints and check constraints. Foreign-key constraints can cause errors, and check constraints can slow down the loading process. Constraints can be reactivated after the load concludes successfully.
- **Indexes:** Remove indexes from the target tables. Indexes are not necessary for inserts. They will slow down the loading process significantly. For each row that is inserted into a table, the database will update every index on that table. You can add back the indexes after the load is finished.

Note:

A primary index is required for all applications that access DB2 for z/OS target tables. You can delete all other indexes from the target tables, except for the primary index.

- **Keys:** For Oracle GoldenGate to reconcile the replicated incremental data changes with the results of the load, each target table must have a primary or unique key. If you cannot create a key through your application, use the `KEYCOLS` option of the `TABLE` and `MAP` parameters to specify columns as a substitute key for Oracle GoldenGate's purposes. A key helps identify which row to process. If you cannot create keys, the source database must be quiesced for the load.

Configure the Manager Process

On the source and target systems, configure and start a Manager process. One Manager can be used for the initial-load processes and the change-synchronization processes. See [Configuring Manager and Network Communications](#) for more information. For enhanced security, the target manager parameter file should have the following parameter for `RMTTASK` to access Replicat on target:

```
ACCESSRULE, PROG *, IPADDR *, ALLOW
```

Create a Data-definitions File

A data-definitions file is required if the source and target databases have dissimilar definitions. Oracle GoldenGate uses this file to convert the data to the format required by the target database. See [Associating Replicated Data with Metadata](#) for more information.

Create Change-synchronization Groups

To prepare for the capture and replication of transactional changes during the initial load, create online Extract and Replicat groups. You will start these groups during the load procedure. See [Configuring Online Change Synchronization](#) for more information.

 **Note:**

If the load is performed from a quiet source database and *will not* be followed by continuous change synchronization, you can omit these groups.

Do not start the Extract or Replicat groups until instructed to do so in the initial-load instructions. Change synchronization keeps track of transactional changes while the load is being applied, and then the target tables are reconciled with those changes.

 **Note:**

The first time that Extract starts in a new Oracle GoldenGate configuration, any open transactions will be skipped. Only transactions that begin after Extract starts are captured.

Sharing Parameters between Process Groups

Some of the parameters that you use in a change-synchronization parameter file also are required in an initial-load Extract and initial-load Replicat parameter file. You can copy those parameters from one parameter file to another, or you can store them in a central file and use the `OBEY` parameter in each parameter file to retrieve them. Alternatively, you can create an Oracle GoldenGate macro for the shared parameters and then call the macro from each parameter file with the `MACRO` parameter.

See [Getting Started with the Oracle GoldenGate Process Interfaces](#) for more information about using `OBEY` and using macros.

Initial Load in Classic Architecture

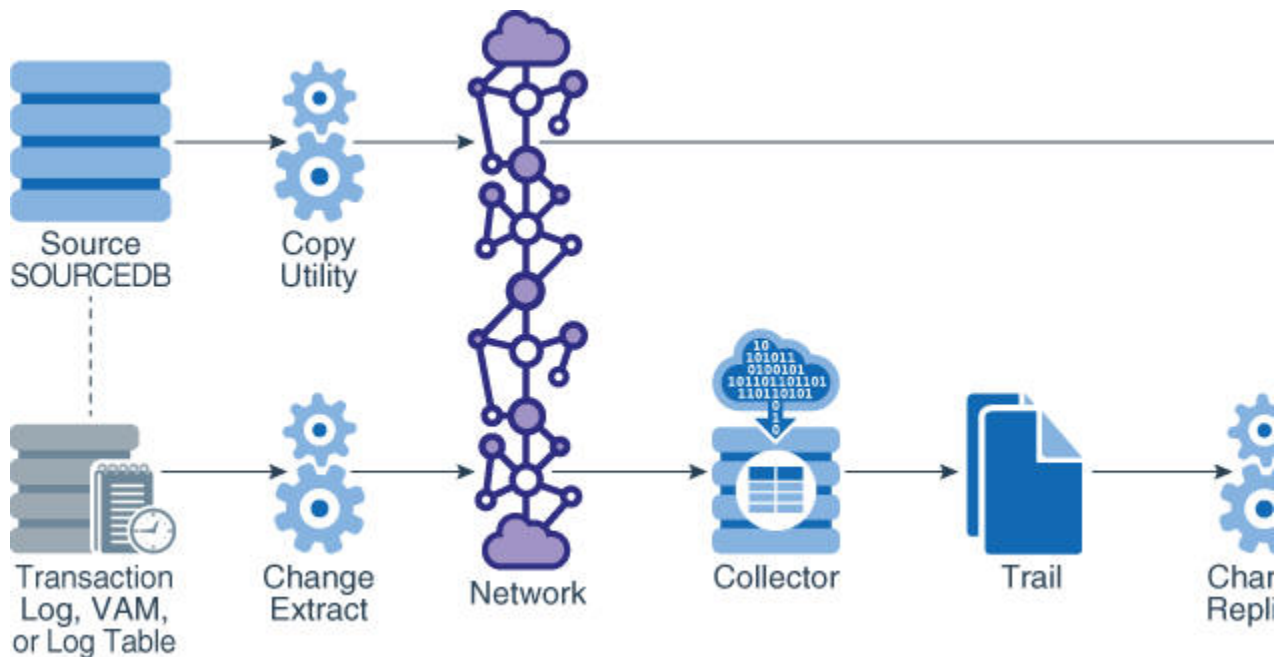
In Classic Architecture you can load data using various options. The processes and steps do so, are described in this topic.

- [Loading Data with a Database Utility](#)
- [Loading Data with Oracle Data Pump](#)

- Loading Data from File to Replicat
- Loading Data with an Oracle GoldenGate Direct Load
- Loading Data with a Direct Bulk Load to SQL*Loader
- Loading Data with Teradata Load Utilities

Loading Data with a Database Utility

To use a database copy utility to establish the target data, you start a change-synchronization Extract group to extract ongoing data changes while the database utility makes and applies a static copy of the data. When the copy is finished, you start the change-synchronization Replicat group to re-synchronize rows that were changed while the copy was being applied. From that point forward, both Extract and Replicat continue running to maintain data synchronization. This method does not involve any special initial-load Extract or Replicat processes.



 **Note:**

The objects and data types being loaded in this method must be supported by Oracle GoldenGate for your database and also by the database utility that is being used. For items that are supported for your database, see the Oracle GoldenGate installation and configuration documentation for that database. For items that are supported by the database utility, see the database vendor's documentation.

1. Make certain that you have addressed the requirements in [Prerequisites for Initial Load](#).

2. On the source and target systems, run GGSCI and start the Manager process.

```
START MANAGER
```

 **Note:**

In a Windows cluster, start the Manager resource from the Cluster Administrator.

3. On the source system, start change extraction.

```
START EXTRACT group
```

Where:

group is the name of the Extract group.

4. (Oracle, if replicating sequences) Issue the `DBLOGIN` command as the user who has `EXECUTE` privilege on `update.Sequence`.

```
GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
```

5. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).

```
FLUSH SEQUENCE owner.sequence
```

6. On the source system, start making the copy.
7. Wait until the copy is finished and record the time of completion.
8. View the Replicat parameter file to make certain that the `HANDLECOLLISIONS` parameter is listed. If not, add the parameter to the file.

 **Caution:**

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

9. On the target system, start change replication.

```
START REPLICAT group
```

Where:

group is the name of the Replicat group.

10. On the target system, issue the following command to verify the status of change replication.

```
INFO REPLICAT group
```


11. Continue to issue the `INFO REPLICAT` command until you have verified that change replication has posted all of the change data that was generated during the initial load. Reference the time of completion that you recorded. For example, if the copy stopped at 12:05, make sure change replication has posted data up to that point.

12. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

```
SEND REPLICAT group, NOHANDLECOLLISIONS
```

13. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

14. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Loading Data with Oracle Data Pump

This method uses the Oracle Data Pump utility to establish the target data. After you apply the copy to the target, you record the SCN at which the copy stopped. Transactions that were included in the copy are skipped to avoid collisions from integrity violations. From the process start point, Oracle GoldenGate maintains data synchronization. No initial-load Oracle GoldenGate processes are required for these methods.

- [Using Automatic Per Table Instantiation](#)
- [Using Oracle Data Pump Table Instantiation](#)

Using Automatic Per Table Instantiation

You can automatically instantiate per table CSN filtering for Oracle Database with Oracle Data Pump, which avoids having all of your the tables at same SCN.

On the Source Database

1. Use `ADD TRANDATA` and `ADD SCHEMATRANDATA` to automatically prepare your tables.
2. Use `INFO TRANDATA` to make sure that your table is prepared for instantiation and at what point it was done.
3. Stop Replicat on the target database.
4. Start Extract with the correct `TABLE` statement.

5. `EXPORT` your tables using Oracle data pump, which automatically generates import actions to set instantiation SCN at the target upon import.

On the Target Database

1. Import your exported tables using Oracle data pump, which populates system tables and views with instantiation SCNs, as well as the specified table data.
2. Start Replicat using one of the following:

For Metadata Trail Replicats, set the `DBOPTIONS ENABLE_INSTANTIATION_FILTERING` parameter in the Replicat parameter file to enable table-level instantiation filtering.

For all other Replicats, set the `DBOPTIONS source_dbase_name global_name` parameter in the Replicat parameter file where `global_name` is the global name of the Oracle source database that the trail is coming from.

Note:

When the source has no `DOMAIN`, do not specify a `DOMAIN` for the downstream database.

Replicat queries the instantiation SCN on any new mapping and filter records accordingly

For more information, see the Reference for Oracle GoldenGate for Windows and UNIX.

Using Oracle Data Pump Table Instantiation

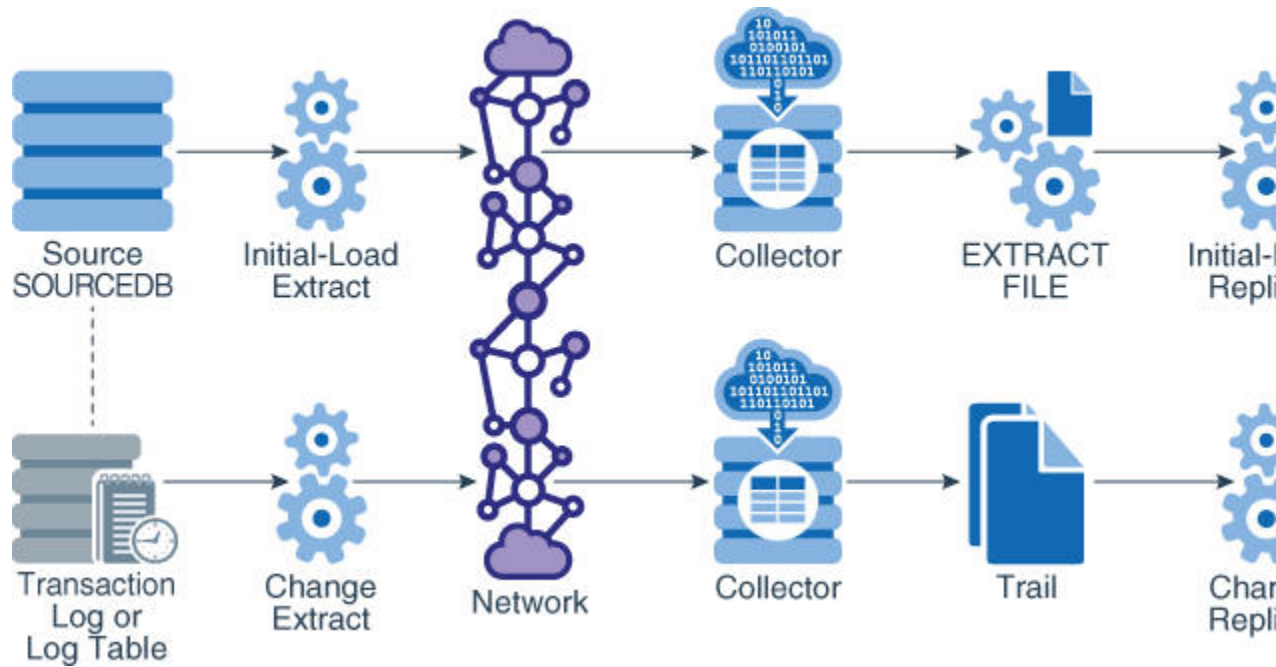
To perform instantiation with Oracle Data Pump, see My Oracle Support document 1276058.1. To obtain this document, do the following:

1. Go to <http://support.oracle.com>.
2. Under Sign In, select your language and then log in with your Oracle Single Sign-On (SSO).
3. On the Dashboard, expand the Knowledge Base heading.
4. Under Enter Search Terms, paste or type the document ID of 1276058.1 and then click **Search**.
5. In the search results, select **Oracle GoldenGate Best Practices: Instantiation from an Oracle Source Database [Article ID 1276058.1]**.
6. Click the link under Attachments to open the article.

Loading Data from File to Replicat

To use Replicat to establish the target data, you use an initial-load Extract to extract source records from the source tables and write them to an extract file in canonical format. From the file, an initial-load Replicat loads the data using the database interface. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.

During the load, the records are applied to the target database one record at a time, so this method is considerably slower than any of the other initial load methods. This method permits data transformation to be done on either the source or target system.



To Load Data From File to Replicat

1. Make certain that you have addressed the requirements in [Prerequisites for Initial Load](#).
2. On the source and target systems, run GGSCI and start Manager.

```
START MANAGER
```

Note:

In a Windows cluster, start the Manager resource from the Cluster Administrator.

3. On the source system, issue the following command to create an initial-load Extract parameter file.

```
EDIT PARAMS initial-load_Extract
```

4. Enter the parameters listed in [Table 15-1](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for loading data from file to Replicat.

```
SOURCEISTABLE
SOURCEDB mydb, USERIDALIAS ogg
RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
ENCRYPTTRAIL AES192
RMTFILE /ggs/dirdat/initld, MEGABYTES 2, PURGE
```

```
TABLE hr.*;
TABLE sales.*;
```

Table 15-1 Initial-Load Extract Parameters

Parameter	Description
SOURCEISTABLE	Designates Extract as an initial load process extracting records directly from the source tables.
SOURCEDB <i>dsn</i> [, USERIDALIAS <i>alias</i> , <i>options</i> , USERID <i>user</i> , <i>options</i>]	Specifies database connection information. SOURCEDB specifies the source data source name (DSN). USERID and USERIDALIAS specify database credentials if required.
RMTHOSTOPTIONS <i>hostname</i> , MGRPORT <i>portnumber</i> [, ENCRYPT <i>algorithm</i> KEYNAME <i>keyname</i>]	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.
ENCRYPTTRAIL <i>algorithm</i>	Encrypts the data in the remote file.
RMTFILE <i>path</i> , [MEGABYTES <i>n</i>]	Specifies the extract file to which the load data will be written. Oracle GoldenGate creates this file during the load. Checkpoints are not maintained with RMTFILE. Note that the size of an extract file cannot exceed 2GB.
<ul style="list-style-type: none"> • <i>path</i> is the relative or fully qualified name of the file. • MEGABYTES designates the size of each file. 	
TABLE <i>container.owner.object</i> ;	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container or catalog unless SOURCECATALOG is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
CATALOGEXCLUDE SCHEMAEXCLUDE TABLEEXCLUDE EXCLUDEWILDCARDOBJECTSONLY	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated TABLE statement. See <i>Reference for Oracle GoldenGate</i> for details.

5. Enter any appropriate optional Extract parameters listed in the *Reference for Oracle GoldenGate*.
6. Save and close the parameter file.
7. On the target system, issue the following command to create an initial-load Replicat parameter file.

```
EDIT PARAMS initial-load_Replicat
```

8. Enter the parameters listed in [Table 15-2](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for loading data from file to Replicat.

```

SPECIALRUN
END RUNTIME
TARGETDB mydb, USERIDALIAS ogg
EXTFILE /ggs/dirdat/initld
SOURCEDEFS /ggs/dirdef/source_defs
MAP hr.*, TARGET hr.*;
MAP sales.*, TARGET hr.*;

```

Table 15-2 Initial-load Replicat parameters

Parameter	Description
SPECIALRUN	Implements the initial-load Replicat as a one-time run that does not use checkpoints.
END RUNTIME	Directs the initial-load Replicat to terminate when the load is finished.
TARGETDB <i>dsn</i> [, USERIDALIAS <i>alias</i> , <i>options</i> , USERID <i>user</i> , <i>options</i>]	Specifies database connection information. TARGETDB specifies the target data source name (DSN). USERID and USERIDALIAS specify database credentials if required.
EXTFILE <i>path</i> • <i>path</i> is the relative or fully qualified name of the file.	Specifies the input extract file specified with the Extract parameter RMTFILE.
{SOURCEDEFS <i>file</i> } ASSUMETARGETDEFS • Use SOURCEDEFS if the source and target tables have different definitions. Specify the relative or fully qualified name of the source-definitions file generated by DEFGEN. • Use ASSUMETARGETDEFS if the source and target tables have the same definitions.	Specifies how to interpret data definitions. For more information about data definitions files, see Associating Replicated Data with Metadata .
SOURCECATALOG	Specifies a default source Oracle container. Enables the use of two-part names (<i>schema.object</i>) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of MAP parameters.

Table 15-2 (Cont.) Initial-load Replicat parameters

Parameter	Description
<pre>MAP container.owner.object, TARGET owner.object[, DEF template] ;</pre>	<p>Specifies a relationship between a source object or objects and a target object or objects. MAP specifies the source object, and TARGET specifies the target object.</p> <p>For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container or catalog unless SOURCECATALOG is used.</p> <p>For the target object, specify only the <i>owner.object</i> components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to load data.</p> <p>See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.</p> <p>The DEF option specifies a definitions template. For more information about data definitions, see Associating Replicated Data with Metadata. See Reference for Oracle GoldenGate for more information and options for the MAP parameter.</p>
<pre>CATALOGEXCLUDE SCHEMAEXCLUDE MAPEXCLUDE EXCLUDEWILDCARDOBJECTSONLY</pre>	<p>Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated MAP statement. See Reference for Oracle GoldenGate for Windows and UNIX for details.</p>

9. Enter any appropriate optional Replicat parameters listed in the *Reference for Oracle GoldenGate*.
10. Save and close the file.
11. View the Replicat parameter file to make certain that the HANDLECOLLISIONS parameter is listed. If not, add the parameter to the file.
12. On the source system, start change extraction.


```
START EXTRACT group
```
13. (Oracle, if replicating sequences) Issue the DBLOGIN command as the user who has EXECUTE privilege on update.Sequence.


```
GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
```
14. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).


```
FLUSH SEQUENCE owner.sequence
```
15. From the directory where Oracle GoldenGate is installed on the source system, start the initial-load Extract.

UNIX and Linux:

```
$ /GGS directory/extract paramfile dirprm/initial-load_Extract.prm  
reportfile path
```

Windows:

```
C:\> GGS directory\extract paramfile dirprm\initial-load_Extract.prm  
reportfile path
```

Where:

initial-load_Extract is the name of the initial-load Extract that you used when creating the parameter file, and *path* is the relative or fully qualified name of the Extract report file.

16. Verify the progress and results of the initial extraction by viewing the Extract report file using the operating system's standard method for viewing files.
17. Wait until the initial extraction is finished.
18. On the target system, start the initial-load Replicat.

UNIX and Linux:

```
$ /GGS directory/replicat paramfile dirprm/initial-load_Replicat.prm  
reportfile path
```

Windows:

```
C:\> GGS directory\replicat paramfile dirprm\initial-load_Replicat.prm  
reportfile path
```

Where:

initial-load_Replicat is the name of the initial-load Replicat that you used when creating the parameter file, and *path* is the relative or fully qualified name of the Replicat report file.

19. When the initial-load Replicat is finished running, verify the results by viewing the Replicat report file using the operating system's standard method for viewing files.
20. On the target system, start change replication.

```
START REPLICAT group
```

21. On the target system, issue the following command to verify the status of change replication.

```
INFO REPLICAT group
```

22. Continue to issue the `INFO REPLICAT` command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.
23. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

```
SEND REPLICAT group, NOHANDLECOLLISIONS
```
24. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

▲ Caution:

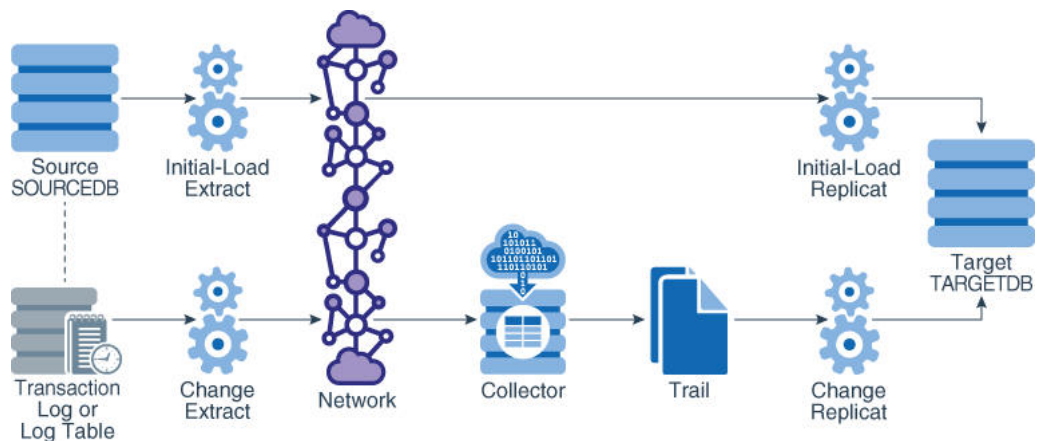
Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

25. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Loading Data with an Oracle GoldenGate Direct Load

To use an Oracle GoldenGate direct load, you run an Oracle GoldenGate initial-load Extract to extract the source records and send them directly to an initial-load Replicat task. A task is started dynamically by the Manager process and does not require the use of a Collector process or file. The initial-load Replicat task delivers the load in large blocks to the target database. Transformation and mapping can be done by Extract, Replicat, or both. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.



To control which port is used by Replicat, and to speed up the search and bind process, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Manager passes the list of port numbers that are specified with this parameter to the Replicat task process. Replicat first searches for a port from this list, and only if no ports are available from the list does Replicat begin scanning in ascending order from the default Manager port number until it finds an available port.

This method supports standard character, numeric, and datetime data types, as well as CLOB, NCLOB, BLOB, LONG, XML, and user-defined datatypes (UDT) embedded with the following attributes: CHAR, NCHAR, VARCHAR, NVARCHAR, RAW, NUMBER, DATE, FLOAT, TIMESTAMP, CLOB, BLOB, XML, and UDT. Character sets are converted between source and target where applicable.

This method supports Oracle internal tables, but does not convert between the source and target character sets during the load.

To Load Data with an Oracle GoldenGate Direct Load

1. Make certain to satisfy "[Prerequisites for Initial Load](#)".
2. On the source and target systems, run GGSCI and start Manager.

```
START MANAGER
```

 **Note:**

In a Windows cluster, start the Manager resource from the Cluster Administrator.

3. On the source, issue the following command to create the initial-load Extract.

```
ADD EXTRACT initial-load_Extract, SOURCEISTABLE
```

Where:

- *initial-load_Extract* is the name of the initial-load Extract, up to eight characters.
- SOURCEISTABLE designates Extract as an initial-load process that reads complete records directly from the source tables. Do not use any of the other ADD EXTRACT service options or datasource arguments.

4. On the source system, issue the following command to create an initial-load Extract parameter file.

```
EDIT PARAMS initial-load_Extract
```

5. Enter the parameters listed in [Table 15-3](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for an Oracle GoldenGate direct load.

```
EXTRACT initext
SOURCEDB mydb, USERIDALIAS ogg
RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
RMTTASK REPLICAT, GROUP initrep
TABLE hr.*;
TABLE sales.*;
```

Table 15-3 Initial-load Extract Parameters for Oracle GoldenGate Direct Load

Parameter	Description
EXTRACT <i>initial-load_Extract</i>	Specifies the initial-load Extract.
SOURCEDB <i>dsn</i> [, USERIDALIAS <i>alias</i> , <i>options</i> , USERID <i>user</i> , <i>options</i>]	Specifies database connection information. SOURCEDB specifies the source datasource name (DSN). See <i>Reference for Oracle GoldenGate</i> for more information. USERID and USERIDALIAS specify database credentials if required.

Table 15-3 (Cont.) Initial-load Extract Parameters for Oracle GoldenGate Direct Load

Parameter	Description
RMTHOSTOPTIONS <i>hostname</i> , MGRPORT <i>portnumber</i> [, ENCRYPT <i>algorithm</i> KEYNAME <i>keyname</i>]	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.
RMTTASK <i>replicat</i> , GROUP <i>initial-load_Replicat</i>	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.
<ul style="list-style-type: none"> <i>initial-load_Replicat</i> is the name of the initial-load Replicat group 	
TABLE <i>container.owner.object</i> ;	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant database, the object name must include the name of the container or catalog unless SOURCECATALOG is used.
CATALOGEXCLUDE SCHEMAEXCLUDE TABLEEXCLUDE EXCLUDEWILDCARDOBJECTSONLY	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated TABLE statement. See <i>Reference for Oracle GoldenGate</i> for details.

- Enter any appropriate optional Extract parameters listed in *Reference for Oracle GoldenGate*.
- Save and close the file.
- On the target system, issue the following command to create the initial-load Replicat task.

```
ADD REPLICAT initial-load_Replicat, SPECIALRUN
```

Where:

- initial-load_Replicat* is the name of the initial-load Replicat task.
 - SPECIALRUN identifies the initial-load Replicat as a one-time run, not a continuous process.
- On the target system, issue the following command to create an initial-load Replicat parameter file.

```
EDIT PARAMS initial-load_Replicat
```


- Enter the parameters listed in [Table 15-4](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for an Oracle GoldenGate direct load.

```
REPLICAT initrep  
TARGETDB mydb, USERIDALIAS ogg  
SOURCEDEFS /ggs/dirdef/source_defs  
MAP hr.*, TARGET hr.*;  
MAP sales.*, TARGET hr.*;
```

Table 15-4 Initial-load Replicat parameters for Oracle GoldenGate Direct Load

Parameter	Description
REPLICAT <i>initial-load_Replicat</i>	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat.
[TARGETDB <i>dsn</i> <i>container</i>] [, USERIDALIAS <i>alias</i> , <i>options</i> , USERID <i>user</i> , <i>options</i>]	Specifies database connection information. TARGETDB specifies the target datasource name (DSN) or Oracle container. See <i>Reference for Oracle GoldenGate</i> for more information. USERID and USERIDALIAS specify database credentials if required.
{SOURCEDEFS <i>full_pathname</i> } ASSUMETARGETDEFS	Specifies how to interpret data definitions. For more information about data definitions files, see Associating Replicated Data with Metadata .
<ul style="list-style-type: none"> Use SOURCEDEFS if the source and target tables have different definitions. Specify the source-definitions file generated by DEFGEN. Use ASSUMETARGETDEFS if the source and target tables have the same definitions. 	
SOURCECATALOG	Specifies a default source Oracle container . Enables the use of two-part names (<i>schema.object</i>) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of MAP parameters.
MAP <i>container.owner.object</i> , TARGET <i>owner.object</i> [, DEF <i>template</i>] ;	Specifies a relationship between a source object or objects and a target object or objects. MAP specifies the source object, and TARGET specifies the target object. For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container or catalog unless SOURCECATALOG is used. For the target object, specify only the <i>owner.object</i> components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to load data. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files. The DEF option specifies a definitions template. For more information about data definitions, see Associating Replicated Data with Metadata . See <i>Reference for Oracle GoldenGate</i> for more information and options for the MAP parameter.

Table 15-4 (Cont.) Initial-load Replicat parameters for Oracle GoldenGate Direct Load

Parameter	Description
CATALOGEXCLUDE	Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated MAP statement. See <i>Reference for Oracle GoldenGate</i> for details.
SCHEMAEXCLUDE	
MAPEXCLUDE	
EXCLUDEWILDCARDOBJECTSONLY	
	<ol style="list-style-type: none"> Enter any appropriate optional Replicat parameters listed in <i>Reference for Oracle GoldenGate</i>. Save and close the parameter file. On the source system, start change extraction. <pre>START EXTRACT group</pre> View the Replicat parameter file to make certain that the <code>HANDLECOLLISIONS</code> parameter is listed. If not, add the parameter to the file. (Oracle, if replicating sequences) Issue the <code>DBLOGIN</code> command as the user who has <code>EXECUTE</code> privilege on <code>update.Sequence</code>. <pre>GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]</pre> (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner). <pre>FLUSH SEQUENCE owner.sequence</pre> On the source system, start the initial-load Extract. <pre>START EXTRACT initial-load_Extract</pre> <div style="border: 1px solid #0070C0; padding: 10px; margin: 10px 0;"> <p> Note:</p> <p>Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.</p> </div> On the target system, issue the following command to find out if the load is finished. Wait until the load is finished before going to the next step. <pre>VIEW REPORT initial-load_Replicat</pre> On the target system, start change replication. <pre>START REPLICAT group</pre> On the target system, issue the following command to verify the status of change replication. <pre>INFO REPLICAT group</pre> Continue to issue the <code>INFO REPLICAT</code> command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.

22. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

```
SEND REPLICAT group, NOHANDLECOLLISIONS
```

23. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

 **Caution:**

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

24. Save and close the parameter file. From this point forward, Oracle GoldenGate continues to synchronize data changes.

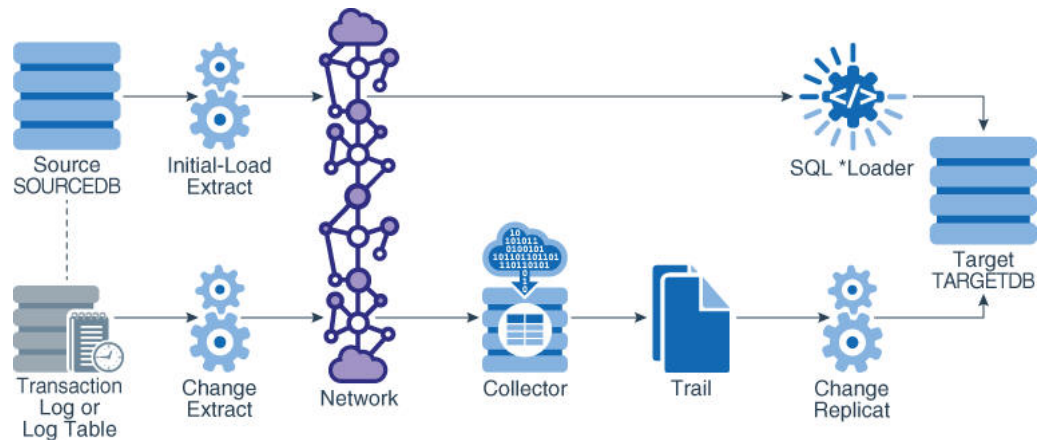
Loading Data with a Direct Bulk Load to SQL*Loader

To use Oracle's SQL*Loader utility to establish the target data, you run an Oracle GoldenGate initial-load Extract to extract the source records and send them directly to an initial-load Replicat task. A task is a process that is started dynamically by the Manager process and does not require the use of a Collector process or file. The initial-load Replicat task interfaces with the API of SQL*Loader to load data as a direct-path bulk load. Data mapping and transformation can be done by either the initial-load Extract or initial-load Replicat, or both. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.

To control which port is used by Replicat, and to speed up the search and bind process, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Manager passes the list of port numbers that are specified with this parameter to the Replicat task process. Replicat first searches for a port from this list, and only if no ports are available from the list does Replicat begin scanning in ascending order from the default Manager port number until it finds an available port.

This method supports standard character, numeric, and datetime data types, as well as `CLOB`, `NCLOB`, `BLOB`, `LONG`, `XML`, and user-defined datatypes (UDT) embedded with the following attributes: `CHAR`, `NCHAR`, `VARCHAR`, `NVARCHAR`, `RAW`, `NUMBER`, `DATE`, `FLOAT`, `TIMESTAMP`, `CLOB`, `BLOB`, `XML`, and `UDT`. `VARRAYS` are not supported. Character sets are converted between source and target where applicable.

This method supports Oracle internal tables, but does not convert between the source and target character sets during the load.



To Load Data With a Direct Bulk Load to SQL*Loader

1. Make certain that you have addressed the requirements in "[Prerequisites for Initial Load](#)".
2. Grant `LOCK ANY TABLE` to the Replicat database user on the target Oracle database.
3. On the source and target systems, run GGSCI and start Manager.

```
START MANAGER
```

4. On the source system, issue the following command to create the initial-load Extract.

```
ADD EXTRACT initial-load_Extract, SOURCEISTABLE
```

Where:

- *initial-load_Extract* is the name of the initial-load Extract, up to eight characters.
 - SOURCEISTABLE designates Extract as an initial-load process that reads complete records directly from the source tables. Do not use any of the other ADD EXTRACT service options or datasource arguments.
5. On the source system, issue the following command to create an initial-load Extract parameter file.

```
EDIT PARAMS initial-load_Extract
```

6. Enter the parameters listed in [Table 15-5](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for a direct bulk load to SQL*Loader.

```
EXTRACT initext
SOURCEDB mydb, USERIDALIAS ogg
RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
RMTTASK REPLICAT, GROUP initrep
TABLE hr.*;
TABLE sales.*;
```

Table 15-5 Initial-load Extract Parameters for a Direct Bulk Load to SQL*Loader

Parameter	Description
EXTRACT <i>initial-load_Extract</i>	Specifies the initial-load Extract.
[, USERIDALIAS <i>alias</i> , <i>options</i> , USERID <i>user</i> , <i>options</i>]	Specifies database connection information. USERID and USERIDALIAS specify database credentials if required.
RMTHOSTOPTIONS <i>hostname</i> , MGRPORT <i>portnumber</i> [, ENCRYPT <i>algorithm</i> KEYNAME <i>keyname</i>]	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.
RMTTASK <i>replicat</i> , GROUP <i>initial-load_Replicat</i>	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.
<ul style="list-style-type: none"> <i>initial-load_Replicat</i> is the name of the initial-load Replicat group. 	
TABLE [<i>container</i> .] <i>owner</i> . <i>object</i> ;	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container unless SOURCECATALOG is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
CATALOGEXCLUDE SCHEMAEXCLUDE TABLEEXCLUDE EXCLUDEWILDCARDOBJECTSONLY	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated TABLE statement. See <i>Reference for Oracle GoldenGate</i> for details.

- Enter any appropriate optional parameters.
- Save and close the file.
- On the target system, issue the following command to create the initial-load Replicat.

```
ADD REPLICAT initial-load_Replicat, SPECIALRUN
```

Where:

- initial-load_Replicat* is the name of the initial-load Replicat task.
 - SPECIALRUN identifies the initial-load Replicat as a one-time task, not a continuous process.
- On the target system, issue the following command to create an initial-load Replicat parameter file.

```
EDIT PARAMS initial-load_Replicat
```

- Enter the parameters listed in [Table 15-6](#) in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for a direct load to SQL*Loader.

```

REPLICAT initrep
USERIDALIAS ogg
BULKLOAD
SOURCEDEFS /ggs/dirdef/source_defs
MAP hr.*, TARGET hr.*;
MAP sales.*, TARGET hr.*;

```

Table 15-6 Initial-load Replicat Parameters for Direct Load to SQL*Loader

Parameter	Description
REPLICAT <i>initial-load_Replicat</i>	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat.
[TARGETDB <i>container</i>] [, USERIDALIAS <i>alias</i> , options , USERID <i>user</i> , options]	Specifies database connection information. TARGETDB specifies the target Oracle container. See <i>Reference for Oracle GoldenGate</i> for more information. USERID and USERIDALIAS specify database credentials if required.
BULKLOAD	Directs Replicat to interface directly with the Oracle SQL*Loader interface. See <i>Reference for Oracle GoldenGate</i> for more information.
{SOURCEDEFS <i>full_pathname</i> } ASSUMETARGETDEFS	Specifies how to interpret data definitions. For more information about data definitions files, see Associating Replicated Data with Metadata . <ul style="list-style-type: none"> Use SOURCEDEFS if the source and target tables have different definitions. Specify the source-definitions file generated by DEFGEN. Use ASSUMETARGETDEFS if the source and target tables have the same definitions.
SOURCECATALOG	Specifies a default source Oracle container for subsequent MAP statements. Enables the use of two-part names (<i>schema.object</i>) where three-part names otherwise would be required. You can use multiple instances of this parameter to specify different default containers for different sets of MAP parameters.

Table 15-6 (Cont.) Initial-load Replicat Parameters for Direct Load to SQL*Loader

Parameter	Description
<pre>MAP [container.]owner.object, TARGET owner.object[, DEF template] ;</pre>	<p>Specifies a relationship between a source object or objects and a target object or objects. MAP specifies the source object, and TARGET specifies the target object.</p> <p>For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container unless SOURCECATALOG is used.</p> <p>For the target object, specify only the <i>owner.object</i> components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container to which you want to load data.</p> <p>See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.</p> <p>The DEF option specifies a definitions template. For more information about data definitions, see Associating Replicated Data with Metadata. See <i>Reference for Oracle GoldenGate</i> for more information and options for the MAP parameter.</p>
<pre>CATALOGEXCLUDE SCHEMAEXCLUDE MAPEXCLUDE EXCLUDEWILDCARDOBJECTSONLY</pre>	<p>Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated MAP statement. See <i>Reference for Oracle GoldenGate</i> for details.</p>

12. Enter any appropriate optional Replicat parameters listed in *Reference for Oracle GoldenGate*.

13. Save and close the parameter file.

14. On the source system, start change extraction.

```
START EXTRACT group
```

15. View the Replicat parameter file to make certain that the HANDLECOLLISIONS parameter is listed. If not, add the parameter to the file.

16. (Oracle, if replicating sequences) Issue the DBLOGIN command as the user who has EXECUTE privilege on update.Sequence.

```
GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
```

17. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).

```
FLUSH SEQUENCE owner.sequence
```

18. On the source system, start the initial-load Extract.

```
START EXTRACT initial-load_Extract
```

▲ Caution:

Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

19. On the target system, issue the following command to determine when the load is finished. Wait until the load is finished before proceeding to the next step.

```
VIEW REPORT initial-load_Extract
```

20. On the target system, start change replication.

```
START REPLICAT group
```

21. On the target system, issue the following command to verify the status of change replication.

```
INFO REPLICAT group
```

22. Continue to issue the `INFO REPLICAT` command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.

23. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

```
SEND REPLICAT group, NOHANDLECOLLISIONS
```

24. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

▲ Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

25. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Loading Data with Teradata Load Utilities

The preferred methods for synchronizing two Teradata databases is to use any of the Teradata data load utilities. The recommended utility is MultiLoad.

This procedure requires Extract and Replicat change-synchronization groups to be available and properly configured for Teradata replication. For more information, see [Configuring Online Change Synchronization](#).

If you are using multiple Extract and Replicat groups, perform each step for all of them as appropriate.

To Load Data With a Teradata Load Utility

1. Create the scripts that are required by the utility.
2. Start the primary Extract group(s).

```
START EXTRACT group
```

3. Start the data pump(s), if used.

```
START EXTRACT data_pump
```

4. Open the Replicat parameter file(s) for editing.

Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

5. Add the following parameters to the Replicat parameter file(s):

```
END RUNTIME  
HANDLECOLLISIONS
```

- `END RUNTIME` directs Replicat to terminate normally when it reads an Oracle GoldenGate trail record that has a timestamp that is the same as, or after, the time that Replicat was started.
 - `HANDLECOLLISIONS` directs Replicat to overwrite duplicate records and ignore missing ones, as a means of resolving errors that occur from collisions between transactional changes and the results of the copy.
6. Save and close the Replicat parameter file(s).
 7. Start the load utility.
 8. When the load completes on the target, start the Replicat process(es).
 9. When each Replicat process stops, remove the `HANDLECOLLISIONS` and `END RUNTIME` parameters from the parameter file.
 10. Restart the Replicat process(es). The two databases are now synchronized, and Oracle GoldenGate will keep them current through replication.

16

Customizing Oracle GoldenGate Processing

This chapter describes how to customize Oracle GoldenGate processing.

Topics:

- [Executing Commands, Stored Procedures, and Queries with SQLEXEC](#)
- [Using Oracle GoldenGate Macros to Simplify and Automate Work](#)
- [Using User Exits to Extend Oracle GoldenGate Capabilities](#)
- [Using the Oracle GoldenGate Event Marker System to Raise Database Events](#)

Executing Commands, Stored Procedures, and Queries with SQLEXEC

The `SQLEXEC` parameter of Oracle GoldenGate enables Extract and Replicat to communicate with the database to do the following:

- Execute a database command, stored procedure, or SQL query to perform a database function, return results (`SELECT` statements) or perform DML (`INSERT`, `UPDATE`, `DELETE`) operations.
- Retrieve output parameters from a procedure for input to a `FILTER` or `COLMAP` clause.

Note:

`SQLEXEC` provides minimal globalization support. To use `SQLEXEC` in the capture parameter file of the source capture, make sure that the client character set in the source `.prm` file is either the same or a superset of the source database character set.

- [Performing Processing with SQLEXEC](#)
- [Using SQLEXEC](#)
- [Executing SQLEXEC within a TABLE or MAP Statement](#)
- [Executing SQLEXEC as a Standalone Statement](#)
- [Using Input and Output Parameters](#)
- [Handling SQLEXEC Errors](#)
- [Additional SQLEXEC Guidelines](#)

Performing Processing with SQLEXEC

SQLEXEC extends the functionality of both Oracle GoldenGate and the database by allowing Oracle GoldenGate to use the native SQL of the database to execute custom processing instructions.

- Stored procedures and queries can be used to select or insert data into the database, to aggregate data, to denormalize or normalize data, or to perform any other function that requires database operations as input. Oracle GoldenGate supports stored procedures that accept input and those that produce output.
- Database commands can be issued to perform database functions required to facilitate Oracle GoldenGate processing, such as disabling triggers on target tables and then enabling them again.

Using SQLEXEC

The SQLEXEC parameter can be used as follows:

- as a clause of a TABLE or MAP statement
- as a standalone parameter at the root level of the Extract or Replicat parameter file.

Executing SQLEXEC within a TABLE or MAP Statement

When used within a TABLE or MAP statement, SQLEXEC can pass and accept parameters. It can be used for procedures and queries, but not for database commands.

Syntax

This syntax executes a procedure within a TABLE or MAP statement.

```
SQLEXEC (SPNAME sp_name,
[ID logical_name,]
{PARAMS param_spec | NOPARAMS})
```

Argument	Description
SPNAME	Required keyword that begins a clause to execute a stored procedure.
<i>sp_name</i>	Specifies the name of the stored procedure to execute.
ID <i>logical_name</i>	Defines a logical name for the procedure. Use this option to execute the procedure multiple times within a TABLE or MAP statement. Not required when executing a procedure only once.
PARAMS <i>param_spec</i> NOPARAMS	Specifies whether or not the procedure accepts parameters. One of these options must be used (see Using Input and Output Parameters).

Syntax

This syntax executes a query within a TABLE or MAP statement.

```
SQLEXEC (ID logical_name, QUERY ' query ',
{PARAMS param_spec | NOPARAMS})
```

Argument	Description
ID <i>logical_name</i>	Defines a logical name for the query. A logical name is required in order to extract values from the query results. ID <i>logical_name</i> references the column values returned by the query.
QUERY ' <i>sql_query</i> '	Specifies the SQL query syntax to execute against the database. It can either return results with a SELECT statement or change the database with an INSERT, UPDATE, or DELETE statement. The query must be within single quotes and must be contained all on one line. Specify case-sensitive object names the way they are stored in the database, such as within quotes for Oracle case-sensitive names. SQLEXEC 'SELECT "col1" from "schema"."table"'
PARAMS <i>param_spec</i> NOPARAMS	Defines whether or not the query accepts parameters. One of these options must be used (see Using Input and Output Parameters).

If you want to execute a query on a table residing on a different database than the current database, then the different database name has to be specified with the table. The delimiter between the database name and the tablename should be a colon (:). The following are some example use cases:

```
select col1 from db1:tab1
select col2 from db2:schema2.tab2
select col3 from tab3
select col3 from schema4.tab4
```

Executing SQLEXEC as a Standalone Statement

When used as a standalone parameter statement in the Extract or Replicat parameter file, SQLEXEC can execute a stored procedure, query, or database command. As such, it need not be tied to any specific table and can be used to perform general SQL operations. For example, if the Oracle GoldenGate database user account is configured to time-out when idle, you could use SQLEXEC to execute a query at a defined interval, so that Oracle GoldenGate does not appear idle. As another example, you could use SQLEXEC to issue an essential database command, such as to disable target triggers. A standalone SQLEXEC statement cannot accept input parameters or return output parameters.

Parameter syntax	Purpose
SQLEXEC 'call <i>procedure_name</i> ()'	Execute a stored procedure
SQLEXEC ' <i>sql_query</i> '	Execute a query
SQLEXEC ' <i>database_command</i> '	Execute a database command

Argument	Description
'call procedure_name ()'	Specifies the name of a stored procedure to execute. The statement must be enclosed within single quotes. Example: SQLEXEC 'call prc_job_count ()'
'sql_query'	Specifies the name of a query to execute. The query must be contained all on one line and enclosed within single quotes. Specify case-sensitive object names the way they are stored in the database, such as within double quotes for Oracle object names that are case-sensitive. SQLEXEC 'SELECT "coll" from "schema"."table"'
'database_command'	Specifies a database command to execute. Must be a valid command for the database.

SQLEXEC provides options to control processing behavior, memory usage, and error handling. For more information, see *Reference for Oracle GoldenGate*.

Using Input and Output Parameters

Oracle GoldenGate provides options for passing input and output values to and from a procedure or query that is executed with SQLEXEC within a TABLE or MAP statement.

- [Passing Values to Input Parameters](#)
- [Passing Values to Output Parameters](#)
- [SQLEXEC Examples Using Parameters](#)

Passing Values to Input Parameters

To pass data values to input parameters within a stored procedure or query, use the PARAMS option of SQLEXEC.

Syntax

```
PARAMS ([OPTIONAL | REQUIRED] param = {source_column | function}
[, ...] )
```

Where:

- OPTIONAL indicates that a parameter value is not required for the SQL to execute. If a required source column is missing from the database operation, or if a column-conversion function cannot complete successfully because a source column is missing, the SQL executes anyway.
- REQUIRED indicates that a parameter value must be present. If the parameter value is not present, the SQL will not be executed.
- param is one of the following:
 - For a stored procedure, it is the name of any parameter in the procedure that can accept input, such as a column in a lookup table.

- For an Oracle query, it is the name of any input parameter in the query excluding the leading colon. For example, `:param1` would be specified as `param1` in the `PARAMS` clause.
- For a non-Oracle query, it is `pn`, where `n` is the number of the parameter within the statement, starting from 1. For example, in a query with two parameters, the `param` entries are `p1` and `p2`.
- `{source_column | function}` is the column or Oracle GoldenGate conversion function that provides input to the procedure.

Passing Values to Output Parameters

To pass values from a stored procedure or query as input to a `FILTER` or `COLMAP` clause, use the following syntax:

Syntax

```
{procedure_name | logical_name}.parameter
```

Where:

- `procedure_name` is the actual name of the stored procedure. Use this argument only if executing a procedure one time during the life of the current Oracle GoldenGate process.
- `logical_name` is the logical name specified with the `ID` option of `SQLEXEC`. Use this argument if executing a query or a stored procedure that will be executed multiple times.
- `parameter` is either the name of the parameter or `RETURN_VALUE`, if extracting returned values.

SQLEXEC Examples Using Parameters

These examples use stored procedures and queries with input and output parameters.

Note:

Additional `SQLEXEC` options are available for use when a procedure or query includes parameters. See the full `SQLEXEC` documentation in *Reference for Oracle GoldenGate*.

Example 16-1 SQLEXEC with a Stored Procedure

This example uses `SQLEXEC` to run a stored procedure named `LOOKUP` that performs a query to return a description based on a code. It then maps the results to a target column named `NEWACCT_VAL`.

```
CREATE OR REPLACE PROCEDURE LOOKUP
(CODE_PARAM IN VARCHAR2, DESC_PARAM OUT VARCHAR2)
BEGIN
  SELECT DESC_COL
  INTO DESC_PARAM
  FROM LOOKUP_TABLE
```



```
WHERE CODE_COL = CODE_PARAM
END;
```

Contents of MAP statement:

```
MAP sales.account, TARGET sales.newacct, &
  SQLEXEC (SPNAME lookup, PARAMS (code_param = account_code)), &
  COLMAP (newacct_id = account_id, newacct_val = lookup.desc_param);
```

SQLEXEC executes the LOOKUP stored procedure. Within the SQLEXEC clause, the PARAMS (code_param = account_code) statement identifies code_param as the procedure parameter to accept input from the account_code column in the account table.

Replicat executes the LOOKUP stored procedure prior to executing the column map, so that the COLMAP clause can extract and map the results to the newacct_val column.

Example 16-2 SQLEXEC with a Query

This example implements the same logic as used in the previous example, but it executes a SQL query instead of a stored procedure and uses the @GETVAL function in the column map.

A query must be on one line. To split an Oracle GoldenGate parameter statement into multiple lines, an ampersand (&) line terminator is required.

Query for an Oracle database:

```
MAP sales.account, TARGET sales.newacct, &
  SQLEXEC (ID lookup, &
  QUERY 'select desc_col desc_param from lookup_table where code_col = :code_param', &
  PARAMS (code_param = account_code)), &
  COLMAP (newacct_id = account_id, newacct_val = &
  @getval (lookup.desc_param));
```

Query for a non-Oracle database:

```
MAP sales.account, TARGET sales.newacct, &
  SQLEXEC (ID lookup, &
  QUERY 'select desc_col desc_param from lookup_table where code_col = ?', &
  PARAMS (p1 = account_code)), &
  COLMAP (newacct_id = account_id, newacct_val = &
  @getval (lookup.desc_param));
```

Handling SQLEXEC Errors

There are two types of error conditions to consider when implementing SQLEXEC:

- The column map requires a column that is missing from the source database operation. This can occur for an update operation if the database only logs the values of columns that changed, rather than all of the column values. By default, when a required column is missing, or when an Oracle GoldenGate column-conversion function results in a "column missing" condition, the stored procedure does not execute. Subsequent attempts to extract an output parameter from the stored procedure results in a "column missing condition" in the COLMAP or FILTER clause.
- The database generates an error.
- [Handling Missing Column Values](#)

- [Handling Database Errors](#)

Handling Missing Column Values

Use the `@COLTEST` function to test the results of the parameter that was passed, and then map an alternative value for the column to compensate for missing values, if desired. Otherwise, to ensure that column values are available, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter to fetch the values from the database if they are not present in the log. As an alternative to fetching columns, you can enable supplemental logging for those columns.

Handling Database Errors

Use the `ERROR` option in the `SQLEXEC` clause to direct Oracle GoldenGate to respond in one of the following ways:

Table 16-1 ERROR Options

Action	Description
IGNORE	Causes Oracle GoldenGate to ignore all errors associated with the stored procedure or query and continue processing. Any resulting parameter extraction results in a "column missing" condition. This is the default.
REPORT	Ensures that all errors associated with the stored procedure or query are reported to the discard file. The report is useful for tracing the cause of the error. It includes both an error description and the value of the parameters passed to and from the procedure or query. Oracle GoldenGate continues processing after reporting the error.
RAISE	Handles errors according to rules set by a <code>REPERROR</code> parameter specified in the Replicat parameter file. Oracle GoldenGate continues processing other stored procedures or queries associated with the current <code>TABLE</code> or <code>MAP</code> statement before processing the error.
FINAL	Performs in a similar way to <code>RAISE</code> except that when an error associated with a procedure or query is encountered, any remaining stored procedures and queries are bypassed. Error processing is called immediately after the error.
FATAL	Causes Oracle GoldenGate to abend immediately upon encountering an error associated with a procedure or query.

Additional SQLEXEC Guidelines

Observe the following `SQLEXEC` guidelines:

- Up to 20 stored procedures or queries can be executed per `TABLE` or `MAP` entry. They execute in the order listed in the parameter statement.
- A database login by the Oracle GoldenGate user must precede the `SQLEXEC` clause. Use the `SOURCEDB` and/or `USERID` or `USERIDALIAS` parameter in the Extract parameter file or the `TARGETDB` and/or `USERID` or `USERIDALIAS` parameter in the Replicat parameter file, as needed for the database type and configured authentication method.
- The SQL is executed by the Oracle GoldenGate user. This user must have the privilege to execute stored procedures and call RDBM-supplied procedures.

- Database operations within a stored procedure or query are committed in same context as the original transaction.
- Do not use `SQLEXEC` to update the value of a primary key column. If `SQLEXEC` is used to update the value of a key column, then the Replicat process will not be able to perform a subsequent update or delete operation, because the original key value will be unavailable. If a key value must be changed, you can map the original key value to another column and then specify that column with the `KEYCOLS` option of the `TABLE` or `MAP` parameter.
- For DB2, Oracle GoldenGate uses the ODBC `SQLExecDirect` function to execute a SQL statement dynamically. This means that the connected database server must be able to prepare the statement dynamically. ODBC prepares the SQL statement every time it is executed (at the requested interval). Typically, this does not present a problem to Oracle GoldenGate users. See the IBM DB2 documentation for more information.
- Do not use `SQLEXEC` for objects being processing by a data-pump Extract in pass-through mode.
- All object names in a `SQLEXEC` statement must be fully qualified with their two-part or three-part names, as appropriate for the database.
- All objects that are affected by a `SQLEXEC` stored procedure or query must exist with the correct structures prior to the execution of the SQL. Consequently, DDL on these objects that affects structure (such as `CREATE` or `ALTER`) must happen before the `SQLEXEC` executes.
- All objects affected by a standalone `SQLEXEC` statement must exist before the Oracle GoldenGate processes start. Because of this, DDL support must be disabled for those objects; otherwise, DDL operations could change the structure or delete the object before the `SQLEXEC` procedure or query executes on it.

Using Oracle GoldenGate Macros to Simplify and Automate Work

You can use Oracle GoldenGate macros in parameter files to configure and reuse parameters, commands, and conversion functions, reducing the amount of text you must enter to do common tasks. A macro is a built-in automation tool that enables you to call a stored set of processing steps from within the Oracle GoldenGate parameter file. A macro can consist of a simple set of frequently used parameter statements to a complex series of parameter substitutions, calculations, or conversions. You can call other macros from a macro. You can store commonly used macros in a library, and then call the library rather than call the macros individually.

Oracle GoldenGate macros work with the following parameter files:

- `DEFGEN`
- `Extract`
- `Replicat`

Do not use macros to manipulate data for tables that are being processed by a data-pump Extract in pass-through mode.

There are two steps to using macros:

[Defining a Macro](#)

Calling a Macro

Topics:

- [Defining a Macro](#)
- [Calling a Macro](#)
- [Calling Other Macros from a Macro](#)
- [Creating Macro Libraries](#)
- [Tracing Macro Expansion](#)

Defining a Macro

To define an Oracle GoldenGate macro, use the `MACRO` parameter in the parameter file. `MACRO` defines any input parameters that are needed and it defines the work that the macro performs.

Syntax

```
MACRO #macro_name
PARAMS (#p1, #p2 [, ...])
BEGIN
macro_body
END;
```

Table 16-2 Macro Definition Arguments

Argument	Description
<code>MACRO</code>	Required. Indicates the start of an Oracle GoldenGate macro definition.
<code>#macro_name</code>	<p>The name of the macro. Macro and parameter names must begin with a macro character. The default macro character is the pound (#) character, as in <code>#macro1</code> and <code>#param1</code>.</p> <p>A macro or parameter name can be one word consisting of letters and numbers, or both. Special characters, such as the underscore character (<code>_</code>) or hyphen (<code>-</code>), can be used. Some examples of macro names are: <code>#mymacro</code>, <code>#macro1</code>, <code>#macro_1</code>, <code>#macro-1</code>, <code>#macro\$</code>. Some examples of parameter names are <code>#sourcecol</code>, <code>#s</code>, <code>#col1</code>, and <code>#col_1</code>.</p> <p>To avoid parsing errors, the macro character cannot be used as the first character of a macro name. For example, <code>##macro</code> is invalid. If needed, you can change the macro character by using the <code>MACROCHAR</code> parameter. See Reference for Oracle GoldenGate for Windows and UNIX.</p> <p>Macro and parameter names are not case-sensitive. Macro or parameter names within quotation marks are ignored.</p>

Table 16-2 (Cont.) Macro Definition Arguments

Argument	Description
PARAMS (#p1, #p2)	Optional definition of input parameters. Specify a comma-separated list of parameter names and enclose it within parentheses. Each parameter must be referenced in the macro body where you want input values to be substituted. You can list each parameter on a separate line to improve readability (making certain to use the open and close parentheses to enclose the parameter list). See Calling a Macro that Contains Parameters for more information.
BEGIN	Begins the macro body. Must be specified before the macro body.
<i>macro_body</i>	The macro body. The body is a syntax statement that defines the function that is to be performed by the macro. A macro body can include any of the following types of statements. <ul style="list-style-type: none"> Simple parameter statements, as in: <pre>COL1 = COL2</pre> Complex parameter statements with parameter substitution as in: <pre>MAP #o.#t, TARGET #o.#t, KEYCOLS (#k), COLMAP (USEDEFAULTS);</pre> Invocations of other macros, as in: <pre>#colmap (COL1, #sourcecol)</pre>
END;	Ends the macro definition. The semicolon is required to complete the definition.

The following is an example of a macro definition that includes parameters. In this case, the macro simplifies the task of object and column mapping by supplying the base syntax of the MAP statement with input parameters that resolve to the names of the owners, the tables, and the KEYCOLS columns.

```
MACRO #macro1
PARAMS ( #o, #t, #k )
BEGIN
MAP #o.#t, TARGET #o.#t, KEYCOLS (#k), COLMAP (USEDEFAULTS);
END;
```

The following is an example of a macro that does not define parameters. It executes a frequently used set of parameters.

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

Calling a Macro

This section shows you how to call a macro. (To define a macro, see [Defining a Macro](#)).

To call a macro, use the following syntax where you want the macro to run within the parameter file.

Syntax

```
[target =] macro_name (val[, ...])
```

```
[target =] macro_name (val | {val, val, ...}[, ...])
```

Table 16-3 Syntax Elements for Calling a Macro

Argument	Description
<code>target =</code>	<p>Optional. Specifies the target to which the results of the macro are assigned or mapped. For example, <code>target</code> can be used to specify a target column in a COLMAP statement. In the following call to the <code>#make_date</code> macro, the column <code>DATECOL1</code> is the target and will be mapped to the macro results.</p> <pre>DATECOL1 = #make_date (YR1, MO1, DAY1)</pre> <p>Without a target, the syntax to call <code>#make_date</code> is:</p> <pre>#make_date (YR1, MO1, DAY1)</pre>
<code>macro_name</code>	The name of the macro that is being called, for example: <code>#make_date</code> .
<code>(val[, ...])</code>	<p>The parameter input values. This component is required whether or not the macro defines parameters. If the macro defines parameters, specify a comma-separated list of input values, in the order that corresponds to the parameter definitions in the MACRO parameter, and enclose the list within parentheses. If the macro does not define parameters, specify the open and close parentheses with nothing between them (). For more information about this syntax, see the following:</p> <ul style="list-style-type: none"> Calling a Macro that Contains Parameters. Calling a Macro without Input Parameters.
<code>(val {val, val, ...})[, ...]</code>	<p>The parameter input values. This component is required whether or not the macro defines parameters. If the macro defines parameters, specify a comma-separated list of input values, in the order that corresponds to the parameter definitions in the MACRO parameter, and enclose the list within parentheses. To pass multiple values to one parameter, separate them with commas and enclose the list within curly brackets. If the macro does not define parameters, specify the open and close parentheses with nothing between them (). For more information about this syntax, see the following:</p> <ul style="list-style-type: none"> Calling a Macro that Contains Parameters. Calling a Macro without Input Parameters.

- [Calling a Macro that Contains Parameters](#)
- [Calling a Macro without Input Parameters](#)

Calling a Macro that Contains Parameters

To call a macro that contains parameters, the call statement must supply the input values that are to be substituted for those parameters when the macro runs. See the syntax in [Table 16-3](#).

Valid input for a macro parameter is any of the following, preceded by the macro character (default is #):

- A single value in plain or quoted text, such as: `#macro (#name, #address, #phone)` OR `#macro ("name", "address", "phone")`.
- A comma-separated list of values enclosed within curly brackets, such as: `#macro1 (SCOTT, DEPT, {DEPTNO1, DEPTNO2, DEPTNO3})`. The ability to substitute a block of values for any given parameter add flexibility to the macro definition and its usability in the Oracle GoldenGate configuration.
- Calls to other macros, such as: `#macro (#mycalc (col2, 100), #total)`. In this example, the `#mycalc` macro is called with the input values of `col2` and `100`.

Oracle GoldenGate substitutes parameter values within the macro body according to the following rules.

1. The macro processor reads through the macro body looking for instances of parameter names specified in the `PARAMS` statement.
2. For each occurrence of the parameter name, the corresponding parameter value specified during the call is substituted.
3. If a parameter name does not appear in the `PARAMS` statement, the macro processor evaluates whether or not the item is, instead, a call to another macro. (See [Calling Other Macros from a Macro](#).) If the call succeeds, the nested macro is executed. If it fails, the whole macro fails.

Example 16-3 Using Parameters to Populate a MAP Statement

The following macro definition specifies three parameter that must be resolved. The parameters substitute for the names of the table owner (parameter `#o`), the table (parameter `#t`), and the `KEYCOLS` columns (parameter `#k`) in a `MAP` statement.

```
MACRO #macro1 PARAMS ( #o, #t, #k ) BEGIN MAP #o.#t, TARGET #o.#t, KEYCOLS (#k),  
COLMAP (USEDEFAULTS); END;
```

Assuming a table in the `MAP` statement requires only one `KEYCOLS` column, the following syntax can be used to call `#macro1`. In this syntax, the `#k` parameter can be resolved with only one value.

```
#macro1 (SCOTT, DEPT, DEPTNO1)
```

To call the macro for a table that requires two `KEYCOLS` columns, the curly brackets are used as follows to enclose both of the required values for the column names:

```
#macro1 (SCOTT, DEPT, {DEPTNO1, DEPTNO2})
```

The `DEPTNO1` and `DEPTNO2` values are passed as one argument to resolve the `#t` parameter. Tables with three or more `KEYCOLS` can also be handled in this manner, using additional values inside the curly brackets.

Example 16-4 Using a Macro to Perform Conversion

In this example, a macro defines the parameters #year, #month, and #day to convert a proprietary date format.

```
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month,
'DD', #day)
END;
```

The macro is called in the COLMAP clause:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcoll = sourcecoll,
datecoll = #make_date(YR1, MO1, DAY1),
datecol2 = #make_date(YR2, MO2, DAY2)
);
```

The macro expands as follows:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcoll = sourcecoll,
datecoll = @DATE ('YYYY-MM-DD', 'CC', @IF (YR1 < 50, 20, 19), 'YY', YR1, 'MM', MO1,
'DD', DAY1),
datecol2 = @DATE ('YYYY-MM-DD', 'CC', @IF (YR2 < 50, 20, 19), 'YY', YR2, 'MM', MO2,
'DD', DAY2)
);
```

Calling a Macro without Input Parameters

To call a macro without input parameters, the call statement must supply the open and close parentheses, but without any input values: #macro ().

The following macro is defined without input parameters. The body contains frequently used parameters.

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

This macro is called as follows:

```
#option_defaults ()
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

#option_defaults ()
MAP owner.srctab2, TARGET owner.targtab2;
```

The macro expands as follows:


```

GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
MAP owner.srctab2, TARGET owner.targtab2;

```

Calling Other Macros from a Macro

To call other macros from a macro, create a macro definition similar to the following. In this example, the `#make_date` macro is nested within the `#assign_date` macro, and it is called when `#assign_date` runs.

The nested macro must define all, or a subset of, the same parameters that are defined in the base macro. In other words, the input values when the base macro is called must resolve to the parameters in both macros.

The following defines `#assign_date`:

```

MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;

```

The following defines `#make_date`. This macro creates a date format that includes a four-digit year, after first determining whether the two-digit input date should be prefixed with a century value of 19 or 20. Notice that the `PARAMS` statement of `#make_date` contains a subset of the parameters in the `#assign_date` macro.

```

MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month,
'DD', #day)
END;

```

The following syntax calls `#assign_date`:

```
#assign_date (COL1, YEAR, MONTH, DAY)
```

The macro expands to the following given the preceding input values and the embedded `#make_date` macro:

```
COL1 = @DATE ('YYYY-MM-DD', 'CC', @IF (YEAR < 50, 20, 19), 'YY', YEAR, 'MM', MONTH,
'DD', DAY)
```

Creating Macro Libraries

You can create a macro library that contains one or more macros. By using a macro library, you can define a macro once and then use it within many parameter files.

To Create a Macro Library

1. Open a new file in a text editor.
2. Use commented lines to describe the library, if needed.
3. Using the syntax described in [Defining a Macro](#), enter the syntax for each macro.
4. Save the file in the `dirprm` sub-directory of the Oracle GoldenGate directory as:

`filename.mac`

Where:

`filename` is the name of the file. The `.mac` extension defines the file as a macro library.

The following sample library named `datelib` contains two macros, `#make_date` and `#assign_date`.

```
-- datelib macro library
--
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month,
'DD', #day)
END;

MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

To use a macro library, use the `INCLUDE` parameter at the beginning of a parameter file, as shown in the following sample Replicat parameter file.

```
INCLUDE /ggs/dirprm/datelib.mac
REPLICAT rep
ASSUMETARGETDEFS
USERIDALIAS ogg
MAP fin.acct_tab, TARGET fin.account;
```

When including a long macro library in a parameter file, you can use the `NOLIST` parameter to suppress the listing of each macro in the Extract or Replicat report file. Listing can be turned on and off by placing the `LIST` and `NOLIST` parameters anywhere within the parameter file or within the macro library file. In the following example, `NOLIST` suppresses the listing of each macro in the `hugelib` macro library. Specifying `LIST` after the `INCLUDE` statement restores normal listing to the report file.

```
NOLIST
INCLUDE /ggs/dirprm/hugelib.mac
LIST
INCLUDE /ggs/dirprm/mdatelib.mac
REPLICAT REP
```

Tracing Macro Expansion

You can trace macro expansion with the `CMDTRACE` parameter. With `CMDTRACE` enabled, macro expansion steps are shown in the Extract or Replicat report file.

Syntax

```
CMDTRACE [ON | OFF | DETAIL]
```

Where:

- `ON` enables tracing.
- `OFF` disables tracing.
- `DETAIL` produces a verbose display of macro expansion.

In the following example, tracing is enabled before `#testmac` is called, then disabled after the macro's execution.

```
REPLICAT REP
MACRO #testmac
BEGIN
COL1 = COL2,
COL3 = COL4,
END;
...
CMDTRACE ON
MAP test.table1, TARGET test.table2,
COLMAP (#testmac);
CMDTRACE OFF
```

Using User Exits to Extend Oracle GoldenGate Capabilities

User exits are custom routines that you write in C programming code and call during Extract or Replicat processing. User exits extend and customize the functionality of the Extract and Replicat processes with minimal complexity and risk. With user exits, you can respond to database events when they occur, without altering production programs.

Topics:

- [When to Implement User Exits](#)
- [Making Oracle GoldenGate Record Information Available to the Routine](#)
- [Creating User Exits](#)
- [Supporting Character-set Conversion in User Exits](#)
- [Using Macros to Check Name Metadata](#)
- [Describing the Character Format](#)
- [Upgrading User Exits](#)
- [Viewing Examples of How to Use the User Exit Functions](#)

When to Implement User Exits

You can employ user exits as an alternative to, or in conjunction with, the column-conversion functions that are available within Oracle GoldenGate. User exits can be a better alternative to the built-in functions because a user exit processes data only once (when the data is extracted) rather than twice (once when the data is extracted and once to perform the transformation).

The following are some ways in which you can implement user exits:

- Perform arithmetic operations, date conversions, or table lookups while mapping from one table to another.
- Implement record archival functions offline.
- Respond to unusual database events in custom ways, for example by sending an e-mail message or a page based on an output value.
- Accumulate totals and gather statistics.
- Manipulate a record.
- Repair invalid data.
- Calculate the net difference in a record before and after an update.
- Accept or reject records for extraction or replication based on complex criteria.
- Normalize a database during conversion.

Making Oracle GoldenGate Record Information Available to the Routine

The basis for most user exit processing is the `EXIT_CALL_PROCESS_RECORD` function. For Extract, this function is called just before a record buffer is output to the trail. For Replicat, it is called just before a record is applied to the target. If source-target mapping is specified in the parameter file, the `EXIT_CALL_PROCESS_RECORD` event takes place after the mapping is performed.

When `EXIT_CALL_PROCESS_RECORD` is called, the record buffer and other record information are available to it through callback routines. The user exit can map, transform, clean, or perform any other operation with the data record. When it is finished, the user exit can return a status indicating whether the record should be processed or ignored by Extract or Replicat.

Creating User Exits

The following instructions help you to create user exits on Windows and UNIX systems. For more information about the parameters and functions that are described in these instructions, see Reference for Oracle GoldenGate for Windows and UNIX.

 **Note:**

User exits are case-sensitive for database object names. Names are returned exactly as they are defined in the hosting database. Object names must be fully qualified.

To Create User Exits

1. In C code, create either a shared object (UNIX systems) or a DLL (Windows) and create or export a routine to be called from Extract or Replicat. This routine is the communication point between Oracle GoldenGate and your routines. Name the routine whatever you want. The routine must accept the following Oracle GoldenGate user exit parameters:
 - `EXIT_CALL_TYPE`: Indicates when, during processing, the routine is called.
 - `EXIT_CALL_RESULT`: Provides a response to the routine.
 - `EXIT_PARAMS`: Supplies information to the routine. This function enables you to use the `EXITPARAM` option of the `TABLE` or `MAP` statement to pass a parameter that is a literal string to the user exit. This is only valid during the exit call to process a specific record. This function also enables you to pass parameters specified with the `PARAMS` option of the `CUSEREXIT` parameter at the exit call startup.
2. In the source code, include the `usrdecs.h` file. The `usrdecs.h` file is the include file for the user exit API. It contains type definitions, return status values, callback function codes, and a number of other definitions. The `usrdecs.h` file is installed within the Oracle GoldenGate directory. Do not modify this file.
3. Include Oracle GoldenGate callback routines in the user exit when applicable. Callback routines retrieve record and application context information, and they modify the contents of data records. To implement a callback routine, use the `ERCALLBACK` function in the shared object. The user callback routine behaves differently based on the function code that is passed to the callback routine.

```
ERCALLBACK (function_code, buffer, result_code);
```

Where:

- `function_code` is the function to be executed by the callback routine.
 - `buffer` is a void pointer to a buffer containing a predefined structure associated with the specified function code.
 - `result_code` is the status of the function that is executed by the callback routine. The result code that is returned by the callback routine indicates whether or not the callback function was successful.
 - On Windows systems, Extract and Replicat export the `ERCALLBACK` function that is to be called from the user exit routine. The user exit must explicitly load the callback function at run-time using the appropriate Windows API calls.
4. Include the `CUSEREXIT` parameter in your Extract or Replicat parameter file. This parameter accepts the name of the shared object or DLL and the name of the exported routine that is to be called from Extract or Replicat. You can specify the full path of the shared object or DLL or let the operating system's standard search strategy locate the shared object.

```
CUSEREXIT {DLL | shared_object} routine
[, INCLUDEUPDATEBEFORES]
[, PARAMS 'startup_string']
```

Where:

- *DLL* is a Windows DLL and *shared_object* is a UNIX shared object that contains the user exit function.
- `INCLUDEUPDATEBEFORES` gets before images for `UPDATE` operations.
- `PARAMS 'startup_string'` supplies a startup string, such as a startup parameter.

Example 16-5 Example of Base Syntax, UNIX

```
CUSEREXIT eruserexit.so MyUserExit
```

Example 16-6 Example Base Syntax, Windows

```
CUSEREXIT eruserexit.dll MyUserExit
```

Supporting Character-set Conversion in User Exits

To maintain data integrity, a user exit needs to understand the character set of the character-type data that it exchanges with an Oracle GoldenGate process. Oracle GoldenGate user exit logic provides globalization support for:

- character-based database metadata, such as the names of catalogs, schemas, tables, and columns
- the values of character-type columns, such as `CHAR`, `VARCHAR2`, `CLOB`, `NCHAR`, `NVARCHAR2`, and `NCLOB`, as well as string-based numbers, date-time, and intervals.

Properly converting between character sets allows column data to be compared, manipulated, converted, and mapped properly from one type of database and character set to another. Most of this processing is performed when the `EXIT_CALL_PROCESS_RECORD` call type is called and the record buffer and other record information is made available through callback routines.

The user exit has its own session character set. This is defined by the `GET_SESSION_CHARSET` and `SET_SESSION_CHARSET` callback functions. The caller process provides conversion between character sets if the character set of the user exit is different from the hosting context of the process.

To enable this support in user exits, there is the `GET_DATABASE_METADATA` callback function code. This function enables the user exit to get database metadata, such as the locale and the character set of the character-type data that it exchanges with the process that calls it (Extract, data pump, Replicat). It also returns how the database treats the case-sensitivity of object names, how it treats quoted and unquoted names, and how it stores object names.

For more information about these components, see Reference for Oracle GoldenGate for Windows and UNIX.

Using Macros to Check Name Metadata

The object name that is passed by the user exit API is the exact name that is encoded in the user-exit session character set, and exactly the same name that is retrieved

from the database. If the user exit compares the object name with a literal string, the user exit must retrieve the database locale and then normalize the string so that it is compared with the object name in the same encoding.

Oracle GoldenGate provides the following macros that can be called by the user exit to check the metadata of database object names. For example, a macro can be used to check whether a quoted table name is case-sensitive and whether it is stored as mixed-case in the database server. These macros are defined in the `usrdecs.h` file.

Table 16-4 Macros for metadata checking

Macro	What it verifies
<code>supportsMixedCaseIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats a mixed-case unquoted name of a specified data type as case-sensitive and stores the name in mixed case.
<code>supportsMixedCaseQuotedIdentifiers(nameMeta, DBObjType)</code>	Whether the database treats the mixed-case quoted name of a specified data type as case-sensitive and stores the name in mixed case.
<code>storesLowerCaseIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in lower case.
<code>storesLowerCaseQuotedIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in lower case.
<code>storesMixedCaseIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in mixed case.
<code>storesMixedCaseQuotedIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in mixed case.
<code>storesUpperCaseIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in upper case.
<code>storesUpperCaseQuotedIdentifiers(nameMeta, DbObjType)</code>	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in upper case.

Describing the Character Format

The input parameter `column_value_mode` describes the character format of the data that is being processed and is used in several of the function codes. The following table describes the meaning of the `EXIT_FN_RAW_FORMAT`, `EXIT_FN_CHAR_FORMAT`, and `EXIT_FN_CNVTED_SESS_FORMAT` format codes, per data type.

Table 16-5 column_value_mode_matrix Meanings

Data Type	EXIT_FN_RAW_FORMAT	EXIT_FN_CHAR_FORMAT	EXIT_FN_CNVTED_SESS_FORMAT
CHAR "abc"	2-byte null indicator + 2-byte length info + column value 0000 0004 61 62 63 20	"abc" encoded in ASCII or EBCDIC. NULL terminated. Trailing spaces are trimmed.	"abc" encoded in user exit session character set. NOT NULL terminated. Trailing spaces are trimmed by default unless the GLOBALS parameter NOTRIMSPACES is specified.
NCHAR 0061 0062 0063 0020	2-byte null indicator + 2-byte length info + column value. 0000 0008 00 61 0062 0063 0020	"abc" (encoded in UTF8) or truncated at the first byte, depending on whether NCHAR is treated as UTF-8. NULL terminated. Trailing spaces are trimmed.	"abc" encoded in user exit session character set. NOT NULL terminated. Trailing spaces are trimmed by default unless the GLOBALS parameter NOTRIMSPACES is specified.
VARCHAR2 "abc"	2-byte null indicator + 2-byte length info + column value	"abc" encoded in ASCII or EBCDIC. NULL terminated. No trimming.	"abc" encoded in user exit session character set. NOT NULL terminated. No trimming.
NVARCHAR2 0061 0062 0063 0020	2-byte null indicator + 2-byte length info + column value	"abc" (encoded in UTF8) or truncated at the first byte, depending on whether NVARCHAR2 is treated as UTF-8. NULL terminated. No trimming.	"abc" encoded in user exit session character set. NOT NULL terminated. No trimming.
CLOB	2-byte null indicator + 2-byte length info + column value	Similar to VARCHAR2, but only output up to 4K bytes. NULL Terminated. No trimming.	Similar to VARCHAR2, but only output data requested in user exit session character set. NOT NULL terminated. No trimming.
NCLOB	2-byte null indicator + 2-byte length info + column value	Similar to NVARCHAR2, but only output up to 4K bytes. NULL terminated. No trimming.	Similar to NVARCHAR2, but only output data requested in user exit session character set. NOT NULL terminated. No trimming.
NUMBER 123.89	2-byte null indicator + 2-byte length info + column value	"123.89" encoded in ASCII or EBCDIC. NULL terminated.	"123.89" encoded in user exit session character set. NOT NULL terminated.
DATE 31-May-11	2-byte null indicator + 2-byte length info + column value	"2011-05-31" encoded in ASCII or EBCDIC. NULL terminated.	"2011-05-31" encoded in user exit session character set. NOT NULL terminated.

Table 16-5 (Cont.) column_value_mode_matrix Meanings

Data Type	EXIT_FN_RAW_FORMAT	EXIT_FN_CHAR_FORMAT	EXIT_FN_CNVTED_SESS_FORMAT
TIMESTAMP 31-May-11 12.00.00 AM	2-byte null indicator + 2-byte length info + column value	"2011-05-31 12.00.00 AM" encoded in ASCII or EBCDIC. NULL terminated.	"2011-05-31 12.00.00 AM" encoded in user exit session character set. NOT NULL terminated.
Interval Year to Month or Interval Day to Second	2-byte null indicator + 2-byte length info + column value	NA	NA
RAW	2-byte null indicator + 2-byte length info + column value	2-byte null indicator + 2-byte length info + column value	2-byte null indicator + 2-byte length info + column value

Upgrading User Exits

The `usrdecs.h` file is versioned to allow backward compatibility with existing user exits when enhancements or upgrades, such as new functions or structural changes, are added to a new Oracle GoldenGate release. The version of the `usrdecs.h` file is printed in the report file at the startup of Replicat or Extract.

To use new user exit functionality, you must recompile your routines to include the new `usrdecs` file. Routines that do not use new features do not need to be recompiled.

Viewing Examples of How to Use the User Exit Functions

Oracle GoldenGate installs the following sample user exit files into the `UserExitExamples` directory of the Oracle GoldenGate installation directory:

- `exitdemo.c` shows how to initialize the user exit, issue callbacks at given exit points, and modify data. It also demonstrates how to retrieve the fully qualified table name or a specific metadata part, such as the name of the catalog or container, or the schema, or just the unqualified table name. In addition, this demo shows how to process DDL data. The demo is not specific to any database type.
- `exitdemo_utf16.c` shows how to use UTF16-encoded data (both metadata and column data) in the callback structures for information exchanged between the user exit and the caller process.
- `exitdemo_more_recs.c` shows an example of how to use the same input record multiple times to generate several target records.
- `exitdemo_lob.c` shows an example of how to get read access to LOB data.
- `exitdemo_pk_befores.c` shows how to access the before and after image portions of a primary key update record, as well as the before images of regular updates (non-key updates). It also shows how to get target row values with `SQLEXEC` in the Replicat parameter file as a means for conflict detection. The resulting fetched values from the target are mapped as the target record when it enters the user exit.

Each directory contains the `*.c` files as well as makefiles and a `readme.txt` file.

Using the Oracle GoldenGate Event Marker System to Raise Database Events

Oracle GoldenGate provides an event marker system, also known as the event marker infrastructure (EMI), which enables the Oracle GoldenGate processes to take a defined action based on an *event record* in the transaction log or in the trail (depending on the data source of the process). The event record is a record that satisfies a specific filter criterion for which you want an action to occur. You can use this system to customize Oracle GoldenGate processing based on database events.

For example, you can use the event marker system to start, suspend, or stop a process, to perform a transformation, or to report statistics. The event marker system can be put to use for purposes such as:

- To establish a synchronization point at which `SQLEXEC` or user exit functions can be performed
- To execute a shell command that executes a data validation script or sends an email
- To activate tracing when a specific account number is detected
- To capture lag history
- To stop or suspend a process to run reports or batch processes at the end of the day

The event marker feature is supported for the replication of data changes, but not for initial loads.

The system requires the following input components:

1. The *event record* that triggers the action can be specified with `FILTER`, `WHERE`, or `SQLEXEC` in a `TABLE` or `MAP` statement. Alternatively, a special `TABLE` statement in a Replicat parameter file enables you to perform `EVENTACTIONS` actions without mapping a source table to a target table.
2. In the `TABLE` or `MAP` statement where you specify the event record, include the `EVENTACTIONS` parameter with the appropriate option to specify the action that is to be taken by the process.

You can combine `EVENTACTIONS` options, as shown in the following examples.

The following causes the process to issue a checkpoint, log an informational message, and ignore the entire transaction (without processing any of it), plus generate a report.

```
EVENTACTIONS (CP BEFORE, REPORT, LOG, IGNORE TRANSACTION)
```

The following writes the event record to the discard file and ignores the entire transaction.

```
EVENTACTIONS (DISCARD, IGNORE TRANS)
```

The following logs an informational message and gracefully stop the process.

```
EVENTACTIONS (LOG INFO, STOP)
```

The following rolls over the trail file and does not write the event record to the new file.

```
EVENTACTIONS (ROLLOVER, IGNORE)
```

For syntax details and additional usage instructions, see *Reference for Oracle GoldenGate*.

- [Case Studies in the Usage of the Event Marker System](#)

Case Studies in the Usage of the Event Marker System

These examples highlight some use cases for the event marker system.

Topics:

- [Trigger End-of-day Processing](#)
- [Simplify Transition from Initial Load to Change Synchronization](#)
- [Stop Processing When Data Anomalies are Encountered](#)
- [Trace a Specific Order Number](#)
- [Execute a Batch Process](#)
- [Propagate Only a SQL Statement without the Resultant Operations](#)
- [Committing Other Transactions Before Starting a Long-running Transaction](#)
- [Execute a Shell Script to Validate Data](#)

Trigger End-of-day Processing

This example specifies the capture of operations that are performed on a special table named `event_table` in the source database. This table exists solely for the purpose of receiving inserts at a predetermined time, for example at 5:00 P.M. every day. When Replicat receives the transaction record for this operation, it stops gracefully to allow operators to start end-of-day processing jobs. By using the insert on the `event_table` table every day, the operators know that Replicat has applied all committed transactions up to 5:00. `IGNORE` causes Replicat to ignore the event record itself, because it has no purpose in the target database. `LOG INFO` causes Replicat to log an informational message about the operation.

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

Simplify Transition from Initial Load to Change Synchronization

Event actions and event tables can be used to help with the transition from an initial load to ongoing change replication. For example, suppose an existing, populated source table must be added to the Oracle GoldenGate configuration. This table must be created on the target, and then the two must be synchronized by using an export/import. This example assumes that an event table named `source.event_table` exists in the source database and is specified in a Replicat `TABLE` statement.

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

To allow users to continue working with the new source table, it is added to the Extract parameter file, but not to the Replicat parameter file. Extract begins capturing data from this table to the trail, where it is stored.

At the point where the source and target are read-consistent after the export, an event record is inserted into the event table on the source, which propagates to the target.

When Replicat receives the event record (marking the read-consistent point), the process stops as directed by `EVENTACTIONS STOP`. This allows the new table to be added to the Replicat `MAP` statement. Replicat can be positioned to start replication from the timestamp of the event record, eliminating the need to use the `HANDLECOLLISIONS` parameter. Operations in the trail from before the event record can be ignored because it is known that they were applied in the export.

The event record itself is ignored by Replicat, but an informational message is logged.

Stop Processing When Data Anomalies are Encountered

This example uses `ABORT` to stop Replicat immediately with a fatal error if an anomaly is detected in a bank record, where the customer withdraws more money than the account contains. In this case, the source table is mapped to a target table in a Replicat `MAP` statement for actual replication to the target. A `TABLE` statement is also used for the source table, so that the `ABORT` action stops Replicat before it applies the anomaly to the target database. `ABORT` takes precedence over processing the record.

```
MAP source.account, TARGET target.account;  
TABLE source.account, FILTER (withdrawal > balance), EVENTACTIONS (ABORT);
```

Trace a Specific Order Number

The following example enables Replicat tracing only for an order transaction that contains an insert operation for a specific order number (`order_no = 1`). The trace information is written to the `order_1.trc` trace file. The `MAP` parameter specifies the mapping of the source table to the target table.

```
MAP sales.order, TARGET rpt.order;  
TABLE source.order,  
FILTER (@GETENV ('GGHEADER', 'OPTYPE') = 'INSERT' AND order_no = 1), &  
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

Execute a Batch Process

In this example, a batch process executes once a month to clear the source database of accumulated data. At the beginning of the transaction, typically a batch transaction, a record is written to a special `job` table to indicate that the batch job is starting. `TRANSACTION` is used with `IGNORE` to specify that the entire transaction must be ignored by Extract, because the target system does not need to reflect the deleted records. By ignoring the work on the Extract side, unnecessary trail and network overhead is eliminated.

```
TABLE source.job, FILTER (@streq (job_type = 'HOUSEKEEPING')=1), &  
EVENTACTIONS (IGNORE TRANSACTION);
```

Note:

If a logical batch delete were to be composed of multiple smaller batches, each smaller batch would require an insert into the job table as the first record in the transaction.

Propagate Only a SQL Statement without the Resultant Operations

This example shows how different `EVENTACTIONS` clauses can be used in combination on the source and target to replicate just a SQL statement rather than the operations that result from that statement. In this case, it is an `INSERT INTO...SELECT` transaction. Such a transaction could generate millions of rows that would need to be propagated, but with this method, all that is propagated is the initial SQL statement to reduce trail and network overhead. The `SELECTs` are all performed on the target. This configuration requires perfectly synchronized source and target tables in order to maintain data integrity.

Extract:

```
TABLE source.statement, EVENTACTIONS (IGNORE TRANS INCLUDEEVENT);
```

Replicat:

```
TABLE source.statement, SQLEXEC (execute SQL statement), &  
EVENTACTIONS (INFO, IGNORE);
```

To use this configuration, a `statement` table is populated with the first operation in the transaction, that being the `INSERT INTO...SELECT`, which becomes the event record.



Note:

For large SQL statements, the statement can be written to multiple columns in the table. For example, eight `VARCHAR (4000)` columns could be used to store SQL statements up to 32 KB in length.

Because of the `IGNORE TRANS INCLUDEEVENT`, Extract ignores all of the subsequent inserts that are associated with the `SELECT` portion of the statement, but writes the event record that contains the SQL text to the trail. Using a `TABLE` statement, Replicat passes the event record to a `SQLEXEC` statement that concatenates the SQL text columns, if necessary, and executes the `INSERT INTO...SELECT` statement using the target tables as the input for the `SELECT` sub-query.

Committing Other Transactions Before Starting a Long-running Transaction

This use of `EVENTACTIONS` ensures that all open transactions that are being processed by Replicat get committed to the target before the start of a long running transaction. It forces Replicat to write a checkpoint before beginning work on the large transaction. Forcing a checkpoint constrains any potential recovery to just the long running transaction. Because a Replicat checkpoint implies a commit to the database, it frees any outstanding locks and makes the pending changes visible to other sessions.

```
TABLE source.batch_table, EVENTACTIONS (CHECKPOINT BEFORE);
```

Execute a Shell Script to Validate Data

This example executes a shell script that runs another script that validates data after Replicat applies the last transaction in a test run. On the source, an event record is written to an event table named `source.event`. The record inserts the value `COMPARE`

into the `event_type` column of the event table, and this record gets replicated at the end of the other test data. In the `TABLE` statement in the Replicat parameter file, the `FILTER` clause qualifies the record and then triggers the shell script `compare_db.sh` to run as specified by `SHELL` in the `EVENTACTIONS` clause. After that, Replicat stops immediately as specified by `FORCESTOP`.

Extract:

```
TABLE src.*;  
TABLE test.event;
```

Replicat:

```
MAP src.*, TARGET targ.*;  
MAP test.event, TARGET test.event, FILTER (@streq (event_type, 'COMPARE')=1), &  
EVENTACTIONS (SHELL 'compare_db.sh', FORCESTOP);
```

17

Monitoring Oracle GoldenGate Processing

This chapter describes the monitoring of Oracle GoldenGate processing.

Topics:

- [Using the Information Commands in GGSCI](#)
- [Monitoring an Extract Recovery](#)
- [Monitoring Lag](#)
- [Using Automatic Heartbeat Tables to Monitor](#)
- [Monitoring Processing Volume](#)
- [Using the Error Log](#)
- [Using the Process Report](#)
- [Using the Discard File](#)
- [Maintaining the Discard and Report Files](#)
- [Reconciling Time Differences](#)
- [Getting Help with Performance Tuning](#)

Using the Information Commands in GGSCI

The primary way to view processing information is through GGSCI. For more information about these commands, see *Reference for Oracle GoldenGate*.

Table 17-1 Commands to View Process Information

Command	What it shows
INFO {EXTRACT REPLICAT} <i>group</i> [DETAIL]	Run status, checkpoints, approximate lag, and environmental information.
INFO MANAGER	Run status and port number
INFO ALL	INFO output for all Oracle GoldenGate processes on the system
STATS {EXTRACT REPLICAT} <i>group</i>	Statistics on processing volume, such as number of operations performed.
STATUS {EXTRACT REPLICAT} <i>group</i>	Run status (starting, running, stopped, abended)
STATUS MANAGER	Run status
LAG {EXTRACT REPLICAT} <i>group</i>	Latency between last record processed and timestamp in the data source
INFO {EXTTRAIL RMTTRAIL} <i>trail</i>	Name of associated process, position of last data processed, maximum file size

Table 17-1 (Cont.) Commands to View Process Information

Command	What it shows
SEND MANAGER	Run status, information about child processes, port information, trail purge settings
SEND {EXTRACT REPLICAT} group	Depending on the process and selected options, returns information about memory pool, lag, TCP statistics, long-running transactions, process status, recovery progress, and more.
VIEW REPORT <i>group</i>	Contents of the discard file or process report
VIEW GGSEVT	Contents of the Oracle GoldenGate error log
COMMAND ER <i>wildcard</i>	Information dependent on the <i>COMMAND</i> type: INFO LAG SEND STATS STATUS <i>wildcard</i> is a wildcard specification for the process groups to be affected, for example: INFO ER <i>ext</i> * STATS ER *
INFO PARAM	Queries for and displays static information.
GETPARAMINFO	Displays currently-running parameter values.

Monitoring an Extract Recovery

If Extract abends when a long-running transaction is open, it can seem to take a long time to recover when it is started again. To recover its processing state, Extract must search back through the online and archived logs (if necessary) to find the first log record for that long-running transaction. The farther back in time that the transaction started, the longer the recovery takes, in general, and Extract can appear to be stalled.

To confirm that Extract is recovering properly, use the `SEND EXTRACT` command with the `STATUS` option. One of the following status notations appears, and you can follow the progress as Extract changes its log read position over the course of the recovery.

- In `recovery[1]` – Extract is recovering to its checkpoint in the transaction log. Meaning that it is reading from either:
 - a) reading from BR checkpoint files and then archived/online logs,
 - or
 - b) reading from Recovery Checkpoint in archived/online log.
- In `recovery[2]` – Extract is recovering from its checkpoint to the end of the trail. Meaning that a recovery marker is appended to the output trail when the last transaction was not completely written then rewriting the transaction.

- `Recovery complete` – The recovery is finished, and normal processing will resume.

Monitoring Lag

Lag statistics show you how well the Oracle GoldenGate processes are keeping pace with the amount of data that is being generated by the business applications. With this information, you can diagnose suspected problems and tune the performance of the Oracle GoldenGate processes to minimize the latency between the source and target databases. See [Tuning the Performance of Oracle GoldenGate](#) for help with tuning Oracle GoldenGate to minimize lag.

Topics:

- [About Lag](#)
- [Controlling How Lag is Reported](#)

About Lag

For Extract, lag is the difference, in seconds, between the time that a record was processed by Extract (based on the system clock) and the timestamp of that record in the data source.

For Replicat, lag is the difference, in seconds, between the time that the last record was processed by Replicat (based on the system clock) and the timestamp of the record in the trail.

To view lag statistics, use either the `LAG` or `SEND` command in GGSCI. For more information, see *Reference for Oracle GoldenGate*.

Note:

The `INFO` command also returns a lag statistic, but this statistic is taken from the last record that was checkpointed, not the current record that is being processed. It is less accurate than `LAG` or `INFO`.

Controlling How Lag is Reported

Use the `LAGREPORTMINUTES` or `LAGREPORHOURS` parameter to specify the interval at which Manager checks for Extract and Replicat lag. See *Reference for Oracle GoldenGate*.

Use the `LAGCRITICALSECONDS`, `LAGCRITICALMINUTES`, or `LAGCRITICALHOURS` parameter to specify a lag threshold that is considered critical, and to force a warning message to the error log when the threshold is reached. This parameter affects Extract and Replicat processes on the local system. See *Reference for Oracle GoldenGate*.

Use the `LAGINFOSECONDS`, `LAGINFOMINUTES`, or `LAGINFOHOURS` parameter to specify a lag threshold; if lag exceeds the specified value, Oracle GoldenGate reports lag information to the error log. If the lag exceeds the value specified with the `LAGCRITICAL` parameter, Manager reports the lag as critical; otherwise, it reports the lag as an informational message. A value of zero (0) forces a message at the frequency

specified with the `LAGREPORTMINUTES` or `LAGREPORHOURS` parameter. See *Reference for Oracle GoldenGate*.

Using Automatic Heartbeat Tables to Monitor

You can use the default automatic heartbeat table functionality to monitor end-to-end replication lag. Automatic heartbeats are sent from each source database into the replication streams, by updating the records in a *heartbeat seed table* and a *heartbeat table*, and constructing a *heartbeat history table*. Each of the replication processes in the replication path process these heartbeat records and update the information in them. These heartbeat records are inserted or updated into the heartbeat table at the target databases.

The heartbeat tables contain the following information:

- Source database
- Destination database
- Information about the outgoing replication streams:
 - Names of the extract, pump/distribution server, and or replicat processes in the path
 - Timestamps when heartbeat records were processed by the replication processes.
- Information about the incoming replication streams:
 - Names of the extract, pump/distribution server, and or replicat processes in the path
 - Timestamps when heartbeat records were processed by the replication processes.

Using the information in the heartbeat table and the heartbeat history table, the current and historical lags in each of the replication can be computed.

In a bidirectional GoldenGate configuration, the heartbeat table has as many entries as the number of replication paths to neighbors that the database has and in a unidirectional setup, the table at the source is empty. The outgoing columns have the timestamps and the outgoing path, the local Extract and the downstream GoldenGate processes. The incoming columns have the timestamps and path of the upstream GoldenGate processes and local replicat.

In a unidirectional configuration, the target database will populate only the incoming columns in the heartbeat table.



Note:

The Automatic Heartbeat functionality is not supported on MySQL version 5.5.

Topics:

- [Understanding Heartbeat Table End-To-End Replication Flow](#)
- [Updating Heartbeat Tables](#)

- [Purging the Heartbeat History Tables](#)
- [Best Practice](#)
- [Using the Automatic Heartbeat Commands](#)

Understanding Heartbeat Table End-To-End Replication Flow

The flow for end-to-end replication as it relates to heartbeat tables relies on the use of Oracle GoldenGate 12.2.0.1 trail format is as follows:

Ensure that Self-Describing Trail Files functionality is enabled, see [Using Self-Describing Trail Files](#).

Enable the heartbeat functionality with the `ENABLE_HEARTBEAT_TABLE` parameter. This is the default.

Add a heartbeat table to each of your databases with the `ADD HEARTBEATTABLE` command. Add the heartbeat table to all source and target instances and then restart existing Oracle GoldenGate processes (not necessary for processes running against HP-OSS for MX) to enable heartbeat functionality. Depending on your specific database system, you may or may not be required to create or enable a job to populate heartbeat table data.

(Optional) For Oracle Databases, you must ensure that the Oracle `DBMS_SCHEDULER` is operating correctly as the heartbeat update relies on it. You can query the `DBMS_SCHEDULER` by issuing:

```
select START_DATE, LAST_START_DATE, NEXT_RUN_DATE
from dba_scheduler_jobs
```

Where `job_name = 'GG_UPDATE_HEARTBEATS'` ;

Then look for valid entries for `NEXT_RUN_DATE`, which is the next time the scheduler will run. If this is a timestamp in the past, then no job will run and you must correct it. A common reason for the scheduler not working is when the parameter `job_queue_processes` is set too low (typically zero). Increase the number of `job_queue_processes` configured in the database with the `ALTER SYSTEM SET JOB_QUEUE_PROCESSES = ##;` command where `##` is the number of job queue processes.

Run an Extract, which on receiving the logical change records (LCR) checks the value in the `OUTGOING_EXTRACT` column.

- If the Extract name matches this value, the `OUTGOING_EXTRACT_TS` column is updated and the record is entered in the trail.
- If the Extract name does not match then the LCR is discarded.
- If the `OUTGOING_EXTRACT` value is `NULL`, it is populated along with `OUTGOING_EXTRACT_TS` and the record is entered in the trail.

The Pump or Distribution server on reading the record, checks the value in the `OUTGOING_ROUTING_PATH` column. This column has a list of distribution paths.

If the value is NULL, the column is updated with the current group name (and path if this is a Distribution server), "*" , update the OUTGOING_ROUTING_TS column, and the record is written into its target trail file.

If the value has a "*" in the list, then replace it with *group name[:pathname]*, "*" , update the OUTGOING_ROUTING_TS column, and the record is written into its target trail file. When the value does not have an asterisk (*) in the list and the pump name is in the list, then the record is sent to the path specified in the relevant *group name[:pathname]*, "*" ' pair in the list. If the pump name is not in the list, the record is discarded.

Run a Replicat, which on receiving the record checks the value in the OUTGOING_REPLICAT column.

- If the Replicat name matches the value, the row in the heartbeat table is updated and the record is inserted into the history table.
- If the Replicat name does not match, the record is discarded.
- If the value is NULL, the row in the heartbeat and heartbeat history tables are updated with an implicit invocation of the Replicat column mapping.

Automatic Replicat Column Mapping:

```

REMOTE_DATABASE      = LOCAL_DATABASE
INCOMING_EXTRACT     = OUTGOING_EXTRACT
INCOMING_ROUTING_PATH = OUTGOING_ROUTING_PATH with "*" removed
INCOMING_REPLICAT    = @GETENV ("GGENVIRONMENT", "GROUPNAME")
INCOMING_HEARTBEAT_TS = HEARTBEAT_TIMESTAMP
INCOMING_EXTRACT_TS  = OUTGOING_EXTRACT_TS
INCOMING_ROUTING_TS  = OUTGOING_ROUTING_TS
INCOMING_REPLICAT_TS = @DATE ('UYYYY-MM-DD
HH:MI:SS.FFFFFFF', 'JTSLCT', @GETENV ('JULIANTIMESTAMP'))
LOCAL_DATABASE      = REMOTE_DATABASE
OUTGOING_EXTRACT     = INCOMING_EXTRACT
OUTGOING_ROUTING_PATH = INCOMING_ROUTING_PATH
OUTGOING_HEARTBEAT_TS = INCOMING_HEARTBEAT_TS
OUTGOING_REPLICAT    = INCOMING_REPLICAT
OUTGOING_HEARTBEAT_TS = INCOMING_HEARTBEAT_TS

```

There is just one column for OUTGOING_ROUTING_TS. If a record passes through multiple pump before being applied by a Replicat, each pump will overwrite the OUTGOING_ROUTING_TS column so that the pumps lag that is calculated is not specific to a single pump and refers to the lag across all the pumps specified in PUMP_PATH.

Additional Considerations:

Computing lags as the heartbeat flows through the system relies on the clocks of the source and target systems to be set up correctly. It is possible that the lag can be negative if the target system is ahead of the source system. The lag is shown as a negative number so that you are aware of their clock discrepancy and can take actions to fix it.

The timestamp that flows through the system is in UTC. There is no time zone associated with the timestamp so when viewing the heartbeat tables, the lag can be viewed quickly even if different components are in different time zones. You can write any view you want on top of the underlying tables; UTC is recommended.

All the heartbeat entries are written to the trail in UTF-8.

The outgoing and incoming paths together uniquely determine a row. Meaning that if you have two rows with same outgoing path and a different incoming path, then it is considered two unique entries.

Heartbeat Table Details

The `GG_HEARTBEAT` table displays timestamp information of the end-to-end replication time and the timing information at the different components primary and secondary Extract and Replicat.

In a unidirectional environment, only the target database contains information about the replication lag. That is the time when a record is generated at the source database and becomes visible to clients at the target database.

Note:

The automatic heartbeat tables don't populate the `OUTGOING_%` columns with data, when both the source and remote databases have the same name. To change the database name, use the utility `DBNEWID`. For details, see the [DBNEWID Utility](#).

Table 17-2 `GG_HEARTBEAT` Table

Column	Data Type	Description
<code>LOCAL_DATABASE</code>	<code>VARCHAR2</code>	Local database where the replication time from the remote database is measured.
<code>HEARTBEAT_TIMESTAMP</code>	<code>TIMESTAMP (6)</code>	The point in time when a timestamp is generated at the remote database.
<code>REMOTE_DATABASE</code>	<code>VARCHAR2</code>	Remote database where the timestamp is generated
<code>INCOMING_EXTRACT</code>	<code>VARCHAR2</code>	Name of the primary Extract (capture) at the remote database
<code>INCOMING_ROUTING_PATH</code>	<code>VARCHAR2</code>	Name of the secondary Extract (pump) at the remote database
<code>INCOMING_REPLICAT</code>	<code>VARCHAR2</code>	Name of the Replicat on the local database.
<code>INCOMING_HEARTBEAT_TS</code>	<code>TIMESTAMP (6)</code>	Final timestamp when the information is inserted into the <code>GG_HEARTBEAT</code> table at the local database.
<code>INCOMING_EXTRACT_TS</code>	<code>TIMESTAMP (6)</code>	Timestamp of the generated timestamp is processed by the primary Extract at the remote database.
<code>INCOMING_ROUTING_TS</code>	<code>TIMESTAMP (6)</code>	Timestamp of the generated timestamp is processed by the secondary Extract at the remote database.

Table 17-2 (Cont.) GG_HEARTBEAT Table

Column	Data Type	Description
INCOMING_REPLICAT_TS	TIMESTAMP (6)	Timestamp of the generated timestamp is processed by Replicat at the local database.
OUTGOING_EXTRACT	VARCHAR2	Bidirectional/N-way replication: Name of the primary Extract on the local database.
OUTGOING_ROUTING_PATH	VARCHAR2	Bidirectional/N-way replication: Name of the secondary Extract on the local database.
OUTGOING_REPLICAT	VARCHAR2	Bidirectional/N-way replication: Name of the Replicat on the remote database.
OUTGOING_HEARTBEAT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Final timestamp when the information is inserted into the table at the remote database.
OUTGOING_EXTRACT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by the primary Extract on the local database.
OUTGOING_ROUTING_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by the secondary Extract on the local database.
OUTGOING_REPLICAT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by Replicat on the remote database.

The GG_HEARTBEAT_HISTORY table displays historical timestamp information of the end-to-end replication time and the timing information at the different components primary and secondary Extract and Replicat.

In a unidirectional environment, only the destination database contains information about the replication lag.

Timestamps are managed in UTC time zone. That is the time when a record is generated at the source database and becomes visible to clients at the target database.

Table 17-3 GG_HEARTBEAT_HISTORY Table

Column	Data Type	Description
LOCAL_DATABASE	VARCHAR2	Local database where the end-to-end lag is measured.
HEARTBEAT_RECEIVED_TS	TIMESTAMP(6)	Point in time when a timestamp from the remote database receives at the local database.
REMOTE_DATABASE	VARCHAR2	Remote database where the timestamp is generated.
INCOMING_EXTRACT	VARCHAR2	Name of the primary Extract on the remote database.
INCOMING_ROUTING_PATH	VARCHAR2	Name of the secondary Extract of the remote database.
INCOMING_REPLICAT	VARCHAR2	Name of the Replicat on the local database.
INCOMING_HEARTBEAT_TS	TIMESTAMP(6)	Final timestamp when the information is inserted into the GG_HEARTBEAT_HISTORY table on the local database.
INCOMING_EXTRACT_TS	TIMESTAMP(6)	Timestamp when the generated timestamp is processed by the primary Extract on the remote database.
INCOMING_ROUTING_TS	TIMESTAMP(6)	Timestamp when the generated timestamp is processed by the secondary Extract on the remote database.
INCOMING_REPLICAT_TS	TIMESTAMP(6)	Timestamp when the generated timestamp is processed by Replicat on the local database.
OUTGOING_EXTRACT	VARCHAR2	Bidirectional/N-way replication: Name of the primary Extract from the local database.
OUTGOING_ROUTING_PATH	VARCHAR2	Bidirectional/N-way replication: Name of the secondary Extract from the local database.
OUTGOING_REPLICAT	VARCHAR2	Bidirectional/N-way replication: Name of the Replicat on the remote database.

Table 17-3 (Cont.) GG_HEARTBEAT_HISTORY Table

Column	Data Type	Description
OUTGOING_HEARTBEAT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Final timestamp when the information is persistently inserted into the table of the remote database.
OUTGOING_EXTRACT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by the primary Extract on the local database.
OUTGOING_ROUTING_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by the secondary Extract on the local database.
OUTGOING_REPLICAT_TS	TIMESTAMP (6)	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by Replicat on the remote database.

The `GG_LAG` view displays information about the replication lag between the local and remote databases.

In a unidirectional environment, only the destination database contains information about the replication lag. The lag is measured in seconds.

Table 17-4 GG_LAG View

Column	Data Type	Description
LOCAL_DATABASE	VARCHAR2	Local database where the end-to-end replication lag from the remote database is measured.
CURRENT_LOCAL_TS	TIMESTAMP (6)	Current timestamp of the local database.
REMOTE_DATABASE	VARCHAR2	Remote database where the timestamp is generated.
INCOMING_HEARTBEAT_AGE	NUMBER	The age of the most recent heartbeat received from the remote database.
INCOMING_PATH	VARCHAR2	Replication path from the remote database to the local database with Extract and Replicat components.

Table 17-4 (Cont.) GG_LAG View

Column	Data Type	Description
INCOMING_LAG	NUMBER	Replication lag from the remote database to the local database. This is the time where the heartbeat where generated at the remote database minus the time where the information was persistently inserted into the table at the local database.
OUTGOING_HEARTBEAT_AGE	NUMBER	The age of the most recent heartbeat from the local database to the remote database.
OUTGOING_PATH	VARCHAR2	Replication Path from Local database to the remote database with Extract and Replicat components
OUTGOING_LAG	NUMBER	Replication Lag from the local database to the remote database. This is the time where the heartbeat where generated at the local database minus the time where the information was persistently inserted into the table at the remote database.

The GG_LAG_HISTORY view displays the history information about the replication lag history between the local and remote databases.

In a unidirectional environment, only the destination database contains information about the replication lag.

The unit of the lag units is in seconds.

Table 17-5 GG_LAG_HISTORY View

Column	Data Type	Description
LOCAL_DATABASE	VARCHAR2	Local database where the end-to-end replication lag from the remote database is measured.
HEARTBEAT_RECEIVED_TS	TIMESTAMP(6)	Point in time when a timestamp from the remote database receives on the local database.
REMOTE_DATABASE	VARCHAR2	Remote database where the timestamp is generated.
INCOMING_HEARTBEAT_AGE	NUMBER	

Table 17-5 (Cont.) GG_LAG_HISTORY View

Column	Data Type	Description
INCOMING_PATH	VARCHAR2	Replication path from the remote database to local database with Extract and Replicat components.
INCOMING_LAG	NUMBER	Replication lag from the remote database to the local database. This is the time where the heartbeat was generated at the remote database minus the time where the information was persistently inserted into the table on the local database.
OUTGOING_HEARTBEAT_AGE	NUMBER	
OUTGOING_PATH	VARCHAR2	Replication path from local database to the remote database with Extract and Replicat components.
OUTGOING_LAG	NUMBER	Replication lag from the local database to the remote database. This is the time where the heartbeat was generated at the local database minus the time where the information was persistently inserted into the table on the remote database.

Updating Heartbeat Tables

The `HEARTBEAT_TIMESTAMP` column in the heartbeat seed table must be updated periodically by a database job. The default heartbeat interval is 1 minute and this interval can be specified or overridden using a GGSCI or administration server command. For Oracle Database, the database job is created automatically; for all other supported databases, you must create background jobs to update the heartbeat timestamp using the database specific scheduler functionality.

Purging the Heartbeat History Tables

The heartbeat history table is purged periodically using a job. The default interval is 30 days and this interval can be specified or overridden using a GGSCI or administration server command. For Oracle Database, the database job is created automatically; for all other supported databases, you must create background jobs to purge the heartbeat history table using the database specific scheduler functionality.

Best Practice

Oracle recommends that you:

- Use the same heartbeat frequency on all the databases to makes diagnosis easier.
- Adjust the retention period if space is an issue.
- Retain the default heartbeat table frequency; the frequency set to be 30 to 60 seconds gives the best results for most workloads.
- Use lag history statistics to collect lag and age information.

Using the Automatic Heartbeat Commands

You can use the heartbeat table commands to control the Oracle GoldenGate automatic heartbeat functionality as follows.

Table 17-6 Heartbeat Table Commands

Command	Description
ADD HEARTBEATTABLE	Creates the objects required for automatic heartbeat functionality.
ALTER HEARTBEATTABLE	Alters existing heartbeat objects.
DELETE HEARTBEATTABLE	Deletes existing heartbeat objects.
DELETE HEARTBEATENTRY	Deletes entries in the heartbeat table.
INFO HEARTBEATTABLE	Displays heartbeat table information.

For more information, see the Reference for Oracle GoldenGate for Windows and UNIX.

Monitoring Processing Volume

The `STATS` commands in GGSCI show you the amount of data that is being processed by an Oracle GoldenGate process, and how fast it is being moved through the Oracle GoldenGate system. With this information, you can diagnose suspected problems and tune the performance of the Oracle GoldenGate processes. These commands provide a variety of options to select and filter the output.

The `STATS` commands are: `STATS EXTRACT`, `STATS REPLICAT`, or `STATS ER` command.

You can send interim statistics to the report file at any time with the `SEND EXTRACT` or `SEND REPLICAT` command with the `REPORT` option.

Using the Error Log

Use the Oracle GoldenGate error log to view:

- a history of GGSCI commands
- Oracle GoldenGate processes that started and stopped
- processing that was performed
- errors that occurred
- informational and warning messages

Because the error log shows events as they occurred in sequence, it is a good tool for detecting the cause (or causes) of an error. For example, you might discover that:

- someone stopped a process
- a process failed to make a TCP/IP or database connection
- a process could not open a file

To view the error log, use any of the following:

- Standard shell command to view the `ggseerr.log` file within the root Oracle GoldenGate directory
- Oracle GoldenGate Director or Oracle GoldenGate Monitor
- `VIEW GGSEVT` command in GGSCI.

Using the Process Report

Use the process report to view (depending on the process):

- parameters in use
- table and column mapping
- database information
- runtime messages and errors
- runtime statistics for the number of operations processed

Every Extract, Replicat, and Manager process generates a report file. The report can help you diagnose problems that occurred during the run, such as invalid mapping syntax, SQL errors, and connection errors.

To view a process report, use any of the following:

- standard shell command for viewing a text file
- Oracle GoldenGate Monitor
- `VIEW REPORT` command in GGSCI.
- To view information if a process abends without generating a report, use the following command to run the process from the command shell of the operating system (not GGSCI) to send the information to the terminal.

```
process paramfile path.prm
```

Where:

- The value for *process* is either `extract` or `replicat`.
- The value for *path.prm* is the fully qualified name of the parameter file, for example:

```
replicat paramfile /ogg/dirdat/repora.prm
```

By default, reports have a file extension of `.rpt`, for example `EXTORA.rpt`. The default location is the `dirrpt` sub-directory of the Oracle GoldenGate directory. However, these properties can be changed when the group is created. Once created, a report file must remain in its original location for Oracle GoldenGate to operate properly after processing has started.

To determine the name and location of a process report, use the `INFO EXTRACT`, `INFO REPLICAT`, or `INFO MANAGER` command in GGSCI.

- [Scheduling Runtime Statistics in the Process Report](#)
- [Viewing Record Counts in the Process Report](#)
- [Preventing SQL Errors from Filling the Replicat Report File](#)

Scheduling Runtime Statistics in the Process Report

By default, runtime statistics are written to the report once, at the end of each run. For long or continuous runs, you can use optional parameters to view these statistics on a regular basis, without waiting for the end of the run.

To set a schedule for reporting runtime statistics, use the `REPORT` parameter in the Extract or Replicat parameter file to specify a day and time to generate runtime statistics in the report. See `REPORT`.

To send runtime statistics to the report on demand, use the `SEND EXTRACT` or `SEND REPLICAT` command with the `REPORT` option to view current runtime statistics when needed.

Viewing Record Counts in the Process Report

Use the `REPORTCOUNT` parameter to report a count of transaction records that Extract or Replicat processed since startup. Each transaction record represents a logical database operation that was performed within a transaction that was captured by Oracle GoldenGate. The record count is printed to the report file and to the screen. For more information, see *Reference for Oracle GoldenGate*.

Preventing SQL Errors from Filling the Replicat Report File

Use the `WARNRATE` parameter to set a threshold for the number of SQL errors that can be tolerated on any target table before being reported to the process report and to the error log. The errors are reported as a warning. If your environment can tolerate a large number of these errors, increasing `WARNRATE` helps to minimize the size of those files. For more information, see *Reference for Oracle GoldenGate*.

Using the Discard File

By default, a discard file is generated whenever a process is started with the `START` command through GGSCI. The discard file captures information about Oracle GoldenGate operations that failed. This information can help you resolve data errors, such as those that involve invalid column mapping.

The discard file reports such information as:

- The database error message
- The sequence number of the data source or trail file
- The relative byte address of the record in the data source or trail file
- The details of the discarded operation, such as column values of a DML statement or the text of a DDL statement.

To view the discard file, use a text editor or use the `VIEW REPORT` command in GGSCI. See *Reference for Oracle GoldenGate*.

The default discard file has the following properties:

- The file is named after the process that creates it, with a default extension of `.dsc`. Example: `finance.dsc`.
- The file is created in the `dirrpt` sub-directory of the Oracle GoldenGate installation directory.
- The maximum file size is 50 megabytes.
- At startup, if a discard file exists, it is purged before new data is written.

You can change these properties by using the `DISCARDFILE` parameter. You can disable the use of a discard file by using the `NODISCARDFILE` parameter. See *Reference for Oracle GoldenGate*.

If a process is started from the command line of the operating system, it does not generate a discard file by default. You can use the `DISCARDFILE` parameter to specify the use of a discard file and its properties.

Once created, a discard file must remain in its original location for Oracle GoldenGate to operate properly after processing has started.

Maintaining the Discard and Report Files

By default, discard files and report files are aged the same way. A new discard or report file is created at the start of a new process run. Old files are aged by appending a sequence number from 0 (the most recent) to 9 (the oldest) to their names.

If the active report or discard file reaches its maximum file size before the end of a run (or over a continuous run), the process abends unless there is an aging schedule in effect. Use the `DISCARDROLLOVER` and `REPORTROLLOVER` parameters to set aging schedules for the discard and report files respectively. These parameters set instructions for rolling over the files at regular intervals, in addition to when the process starts. Not only does this control the size of the files and prevent process outages, but it also provides a predictable set of archives that can be included in your archiving routine. For more information, see the following documentation:

- `DISCARDROLLOVER`
- `REPORTROLLOVER`

No process ever has more than ten aged reports or discard files and one active report or discard file. After the tenth aged file, the oldest is deleted when a new report is created. It is recommended that you establish an archiving schedule for aged reports and discard files in case they are needed to resolve a service request.

Table 17-7 Current Extract and Manager Reports Plus Aged Reports

Permissions	X	Date	Report
-rw-rw-rw-	1 ggs ggs	1193 Oct 11 14:59	MGR.rpt
-rw-rw-rw-	1 ggs ggs	3996 Oct 5 14:02	MGR0.rpt

Table 17-7 (Cont.) Current Extract and Manager Reports Plus Aged Reports

Permissions	X	Date	Report
-rw-rw-rw-	1 ggs ggs	4384 Oct 5 14:02	TCUST.rpt
-rw-rw-rw-	1 ggs ggs	1011 Sep 27 14:10	TCUST0.rpt
-rw-rw-rw-	1 ggs ggs	3184 Sep 27 14:10	TCUST1.rpt
-rw-rw-rw-	1 ggs ggs	2655 Sep 27 14:06	TCUST2.rpt
-rw-rw-rw-	1 ggs ggs	2655 Sep 27 14:04	TCUST3.rpt
-rw-rw-rw-	1 ggs ggs	2744 Sep 27 13:56	TCUST4.rpt
-rw-rw-rw-	1 ggs ggs	3571 Aug 29 14:27	TCUST5.rpt

Reconciling Time Differences

To account for time differences between source and target systems, use the `TCPSOURCETIMER` parameter in the Extract parameter file. This parameter adjusts the timestamps of replicated records for reporting purposes, making it easier to interpret synchronization lag. For more information, see *Reference for Oracle GoldenGate*.

Getting Help with Performance Tuning

See [Tuning the Performance of Oracle GoldenGate](#) for help with tuning the performance of Oracle GoldenGate.

18

Tuning the Performance of Oracle GoldenGate

This chapter contains suggestions for improving the performance of Oracle GoldenGate components.

Topics:

- [Using Multiple Process Groups](#)
- [Splitting Large Tables Into Row Ranges Across Process Groups](#)
- [Configuring Oracle GoldenGate to Use the Network Efficiently](#)
- [Eliminating Disk I/O Bottlenecks](#)
- [Managing Virtual Memory and Paging](#)
- [Optimizing Data Filtering and Conversion](#)
- [Tuning Replicat Transactions](#)

Using Multiple Process Groups

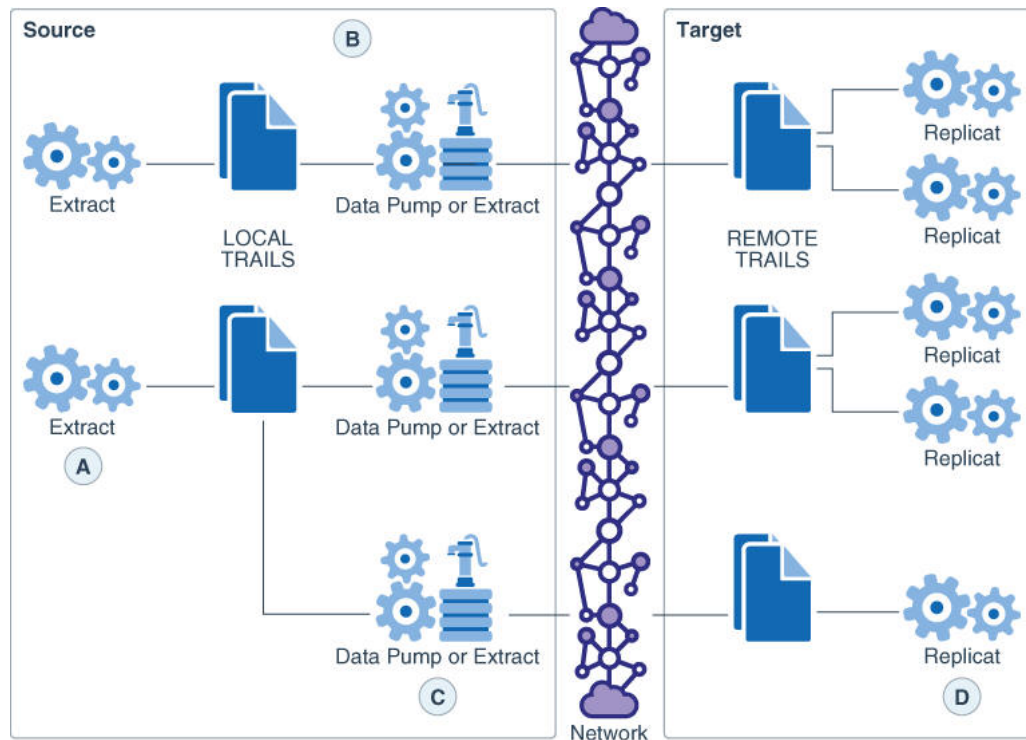
Typically, only one Extract group is required to efficiently capture from a database. However, depending on the redo (transactional) values, or the data and operation types, you may find that you are required to add one or more Extract group to the configuration.

Similarly, only one Replicat group is typically needed to apply data to a target database if using Replicat in coordinated mode. (See [About Coordinated Replicat Mode](#) for more information.) However, even in some cases when using Replicat in coordinated mode, you may be required to use multiple Replicat groups. If you are using Replicat in classic mode and your applications generate a high transaction volume, you probably will need to use parallel Replicat groups.

Because each Oracle GoldenGate component — Extract, data pump, trail, Replicat — is an independent module, you can combine them in ways that suit your needs. You can use multiple trails and parallel Extract and Replicat processes (with or without data pumps) to handle large transaction volume, improve performance, eliminate bottlenecks, reduce latency, or isolate the processing of specific data.

[Figure 18-1](#) shows some of the ways that you can configure Oracle GoldenGate to improve throughput speed and overcome network bandwidth issues.

Figure 18-1 Load-balancing configurations that improve performance



The image labels imply the following:

- **A:** Parallel Extracts divide the load. For example, by schema or to isolate tables that generate fetches.
- **B:** A data pump with local trail can be used for filtering, conversion, and network false tolerance.
- **C:** Multiple data pumps work around network per-process bandwidth limitations to enable TCP/IP throughput. Divide the TABLE parameter statements among them.
- **D:** Parallel Replicats increase throughput to the database. Any trail can be read by one or more Replicats. Divide MAP statements among them.
- [Considerations for Using Multiple Process Groups](#)
- [Using Parallel Replicat Groups on a Target System](#)
- [Using Multiple Extract Groups with Multiple Replicat Groups](#)

Considerations for Using Multiple Process Groups

Before configuring multiple processing groups, review the following considerations to ensure that your configuration produces the desired results and maintains data integrity.

- [Maintaining Data Integrity](#)
- [Number of Groups](#)
- [Memory](#)

- [Isolating Processing-Intensive Tables](#)

Maintaining Data Integrity

Not all workloads can be partitioned across multiple groups and still preserve the original transaction atomicity. You must determine whether the objects in one group will ever have dependencies on objects in any other group, transactional or otherwise. For example, tables for which the workload routinely updates the primary key cannot easily be partitioned in this manner. DDL replication (if supported for the database) is not viable in this mode, nor is the use of some `SQLEXEC` or `EVENTACTIONS` features that base their actions on a specific record.

If your tables do not have any foreign- key dependencies or updates to primary keys, you may be able to use multiple processes. Keep related DML together in the same process stream to ensure data integrity.

Number of Groups

The number of concurrent Extract and Replicat process groups that can run on a system depends on how much system memory is available. Each Classic Extract and Replicat process needs approximately 25-55 MB of memory or more, depending on the size of the transactions and the number of concurrent transactions. The Oracle GoldenGate GGSCI command interface fully supports up to 5,000 concurrent Extract and Replicat groups (combined) per instance of Oracle GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the `INFO` and `STATUS` commands. Beyond the supported level, group information is not displayed and errors may occur. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at a more manageable level, such as 100 or below, in order to manage your environment effectively. The maximum number of groups is controlled by the `MAXGROUPS` parameter, whose default is 1000.

For Windows Server environments, the number of process groups that can be run are tightly coupled to the 'non-interactive' Windows desktop heap memory settings. The default settings for Windows desktop heap may be enough to run very small numbers of process groups, but as you approach larger amounts of process groups, more than 60 or so, you will either need to adjust the 'non-interactive' value of the `SharedSection` field in the registry, based on this information from Microsoft (Windows desktop heap memory), or increase the number of Oracle GoldenGate homes and spread the total number of desired process groups across these homes.

 **Note:**

For more information on modifying the Windows Desktop Heap memory, review the following Oracle Knowledge Base document (Doc ID 2056225.1).

Memory

The system must have sufficient swap space for each Oracle GoldenGate Extract and Replicat process that will be running. To determine the required swap space:

1. Start up one Extract or Replicat.
2. Run GGSCI.

3. View the report file and find the line `PROCESS VM AVAIL FROM OS (min)`.
4. Round up the value to the next full gigabyte if needed. For example, round up 1.76GB to 2 GB.
5. Multiply that value by the number of Extract and Replicat processes that will be running. The result is the maximum amount of swap space that could be required

See the `CACHEMGR` parameter in *Reference for Oracle GoldenGate* for more information about how memory is managed.

Isolating Processing-Intensive Tables

You can use multiple process groups to support certain kinds of tables that tend to interfere with normal processing and cause latency to build on the target. For example:

- Extract may need to perform a fetch from the database because of the data type of the column, because of parameter specifications, or to perform SQL procedures. When data must be fetched from the database, it affects the performance of Extract. You can get fetch statistics from the `STATS EXTRACT` command if you include the `STATOPTIONS REPORTFETCH` parameter in the Extract parameter file. You can then isolate those tables into their own Extract groups, assuming that transactional integrity can be maintained.
- In its classic mode, Replicat process can be a source of performance bottlenecks because it is a single-threaded process that applies operations one at a time by using regular SQL. Even with `BATCHSQL` enabled (see *Reference for Oracle GoldenGate*) Replicat may take longer to process tables that have large or long-running transactions, heavy volume, a very large number of columns that change, and LOB data. You can then isolate those tables into their own Replicat groups, assuming that transactional integrity can be maintained.

Using Parallel Replicat Groups on a Target System

This section contains instructions for creating a configuration that pairs one Extract group with multiple Replicat groups. Although it is possible for multiple Replicat processes to read a single trail (no more than three of them to avoid disk contention) it is recommended that you pair each Replicat with its own trail and corresponding Extract process.

- Refer to *Reference for Oracle GoldenGate for Windows and UNIX* for command and parameter syntax.
- For detailed instructions on configuring change synchronization, see [Configuring Online Change Synchronization](#).
- [To Create the Extract Group](#)
- [To Create the Replicat Groups](#)

To Create the Extract Group



Note:

This configuration includes Extract data-pumps.

1. On the source, use the `ADD EXTRACT` command to create a primary Extract group.
2. On the source, use the `ADD EXTTRAIL` command to specify as many local trails as the number of Replicat groups that you will be creating. All trails must be associated with the primary Extract group.
3. On the source create a data-pump Extract group.
4. On the source, use the `ADD RMTTRAIL` command to specify as many remote trails as the number of Replicat groups that you will be creating. All trails must be associated with the data-pump Extract group.
5. On the source, use the `EDIT PARAMS` command to create Extract parameter files, one for the primary Extract and one for the data pump, that contain the parameters required for your database environment. When configuring Extract, do the following:
 - Divide the source tables among different `TABLE` parameters.
 - Link each `TABLE` statement to a different trail. This is done by placing the `TABLE` statements after the `EXTTRAIL` or `RMTTRAIL` parameter that specifies the trail you want those statements to be associated with.

To Create the Replicat Groups

1. On the target, create a Replicat checkpoint table. For instructions, see [Creating a Checkpoint Table](#). All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each trail that you created. Use the `EXTTRAIL` argument of `ADD REPLICAT` to link the Replicat group to the appropriate trail.
3. On the target, use the `EDIT PARAMS` command to create a Replicat parameter file for each Replicat group that contains the parameters required for your database environment. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to that group.
4. In the Manager parameter file on the target system, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trails.

Using Multiple Extract Groups with Multiple Replicat Groups

Multiple Extract groups write to their own trails. Each trail is read by a dedicated Replicat group.

- Refer to Reference for Oracle GoldenGate for Windows and UNIX for command and parameter syntax.
- For detailed instructions on configuring change synchronization, see [Configuring Online Change Synchronization](#).
- [To Create the Extract Groups](#)
- [To Create the Replicat Groups](#)

To Create the Extract Groups



Note:

This configuration includes data pumps.

1. On the source, use the `ADD EXTRACT` command to create the primary Extract groups.
2. On the source, use the `ADD EXTTRAIL` command to specify a local trail for each of the Extract groups that you created.
3. On the source create a data-pump Extract group to read each local trail that you created.
4. On the source, use the `ADD RMTTRAIL` command to specify a remote trail for each of the data-pumps that you created.
5. On the source, use the `EDIT PARAMS` command to create an Extract parameter file for each primary Extract group and each data-pump Extract group.

To Create the Replicat Groups

1. On the target, create a Replicat checkpoint table. For instructions, see [Creating a Checkpoint Table](#). All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each trail. Use the `EXTTRAIL` argument of `ADD REPLICAT` to link the group to the trail.
3. On the target, use the `EDIT PARAMS` command to create a Replicat parameter file for each Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.
4. In the Manager parameter files on the source system and the target system, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trails.

Splitting Large Tables Into Row Ranges Across Process Groups

You can use the `@RANGE` function to divide the rows of any table across two or more Oracle GoldenGate processes. It can be used to increase the throughput of large and heavily accessed tables and also can be used to divide data into sets for distribution to different destinations. Specify each range in a `FILTER` clause in a `TABLE` or `MAP` statement.

`@RANGE` is safe and scalable. It preserves data integrity by guaranteeing that the same row will always be processed by the same process group.

It might be more efficient to use the primary Extract or a data pump to calculate the ranges than to use Replicat. To calculate ranges, Replicat must filter through the entire trail to find data that meets the range specification. However, your business case should determine where this filtering is performed.

Figure 18-2 Dividing rows of a table between two Extract groups

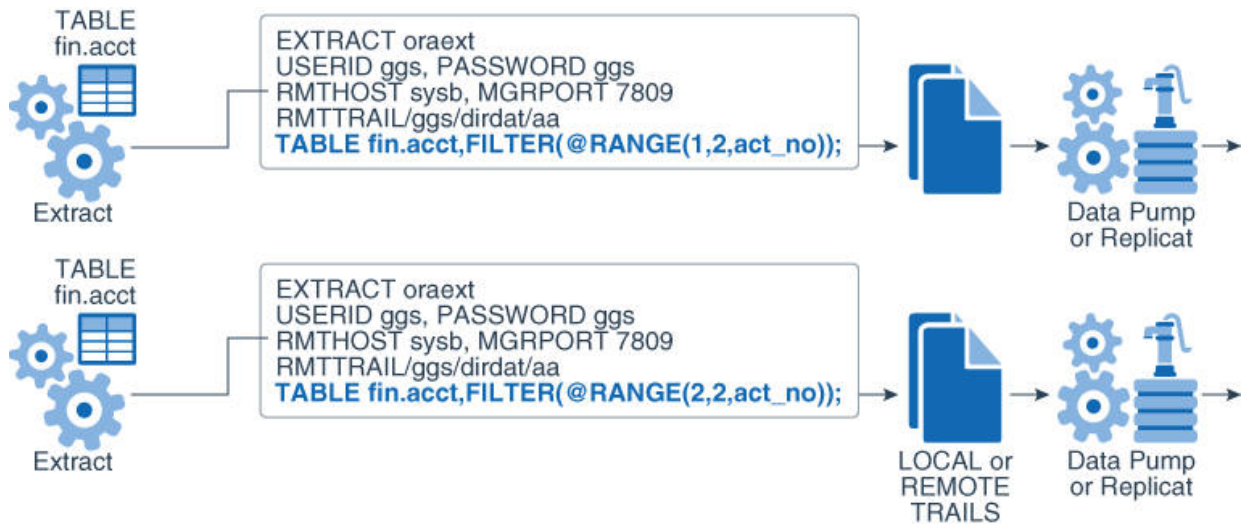
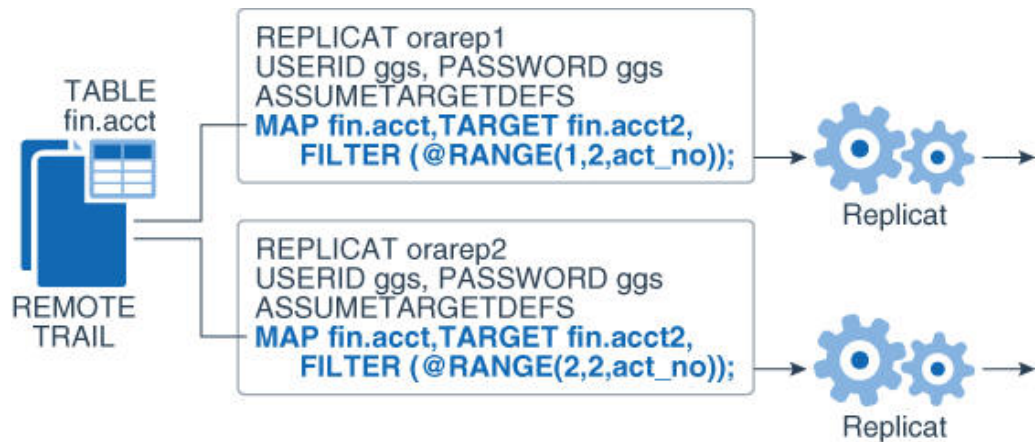


Figure 18-3 Dividing rows of a table between two Replicat groups



Configuring Oracle GoldenGate to Use the Network Efficiently

Inefficiencies in the transfer of data across the network can cause lag in the Extract process and latency on the target. If not corrected, it can eventually cause process failures.

When you first start a new Oracle GoldenGate configuration:

1. Establish benchmarks for what you consider to be acceptable lag and throughput volume for Extract and for Replicat. Keep in mind that Extract will normally be faster than Replicat because of the kind of tasks that each one performs. Over time you will know whether the difference is normal or one that requires tuning or troubleshooting.

2. Set a regular schedule to monitor those processes for lag and volume, as compared to the benchmarks. Look for lag that remains constant or is growing, as opposed to occasional spikes. Continuous, excess lag indicates a bottleneck somewhere in the Oracle GoldenGate configuration. It is a critical first indicator that Oracle GoldenGate needs tuning or that there is an error condition.

To view volume statistics, use the `STATS EXTRACT` or `STATS REPLICAT` command. To view lag statistics, use the `LAG EXTRACT` or `LAG REPLICAT` command. See Reference for Oracle GoldenGate for Windows and UNIX for more information.

- [Detecting a Network Bottleneck that is Affecting Oracle GoldenGate](#)
- [Working Around Bandwidth Limitations by Using Data Pumps](#)
- [Reducing the Bandwidth Requirements of Oracle GoldenGate](#)
- [Increasing the TCP/IP Packet Size](#)

Detecting a Network Bottleneck that is Affecting Oracle GoldenGate

To detect a network bottleneck that is affecting the throughput of Oracle GoldenGate, follow these steps.

1. Issue the following command to view the ten most recent Extract checkpoints. If you are using a data-pump Extract on the source system, issue the command for the primary Extract and also for the data pump.

```
INFO EXTRACT group, SHOWCH 10
```

2. Look for the `Write Checkpoint` statistic. This is the place where Extract is writing to the trail.

```
Write Checkpoint #1
```

```
GG5 Log Trail
```

```
Current Checkpoint (current write position):
```

```
Sequence #: 2
RBA: 2142224
Timestamp: 2011-01-09 14:16:50.567638
Extract Trail: ./dirdat/eh
```

3. For both the primary Extract and data pump:
 - Determine whether there are more than one or two checkpoints. There can be up to ten.
 - Find the `Write Checkpoint n` heading that has the highest increment number (for example, `Write Checkpoint #8`) and make a note of the `Sequence`, `RBA`, and `Timestamp` values. This is the most recent checkpoint.
4. Refer to the information that you noted, and make the following validation:
 - Is the primary Extract generating a series of checkpoints, or just the initial checkpoint?
 - If a data pump is in use, is it generating a series of checkpoints, or just one?
5. Issue `INFO EXTRACT` for the primary and data pump Extract processes again.
 - Has the most recent write checkpoint increased? Look at the most recent `Sequence`, `RBA`, and `Timestamp` values to see if their values were incremented forward since the previous `INFO EXTRACT` command.
6. Issue the following command to view the status of the Replicat process.


```
SEND REPLICAT group, STATUS
```

- The status indicates whether Replicat is delaying (waiting for data to process), processing data, or at the end of the trail (EOF).

There is a network bottleneck if the status of Replicat is either in delay mode or at the end of the trail file and either of the following is true:

- You are only using a primary Extract and its write checkpoint is not increasing or is increasing too slowly. Because this Extract process is responsible for sending data across the network, it will eventually run out of memory to contain the backlog of extracted data and abend.
- You are using a data pump, and its write checkpoint is not increasing, but the write checkpoint of the primary Extract is increasing. In this case, the primary Extract can write to its local trail, but the data pump cannot write to the remote trail. The data pump will abend when it runs out of memory to contain the backlog of extracted data. The primary Extract will run until it reaches the last file in the trail sequence and will abend because it cannot make a checkpoint.

 **Note:**

Even when there is a network outage, Replicat will process in a normal manner until it applies all of the remaining data from the trail to the target. Eventually, it will report that it reached the end of the trail file.

Working Around Bandwidth Limitations by Using Data Pumps

Using parallel data pumps may enable you to work around bandwidth limitations that are imposed on a per-process basis in the network configuration. You can use parallel data pumps to send data to the same target system or to different target systems. Data pumps also remove TCP/IP responsibilities from the primary Extract, and their local trails provide fault tolerance.

Reducing the Bandwidth Requirements of Oracle GoldenGate

Use the compression options of the `RMTHOST` parameter to compress data before it is sent across the network. Weigh the benefits of compression against the CPU resources that are required to perform the compression. See *Reference for Oracle GoldenGate* for more information.

Increasing the TCP/IP Packet Size

Use the `TCPBUFSIZE` option of the `RMTHOST` parameter to control the size of the TCP socket buffer that Extract maintains. By increasing the size of the buffer, you can send larger packets to the target system. See *Reference for Oracle GoldenGate* for more information.

Use the following steps as a guideline to determine the optimum buffer size for your network.

1. Use the `ping` command from the command shell obtain the average round trip time (RTT), shown in the following example:


```
C:\home\ggs>ping ggsoftware.com
Pinging ggsoftware.com [192.168.116.171] with 32 bytes of data:
Reply from 192.168.116.171: bytes=32 time=31ms TTL=56
Reply from 192.168.116.171: bytes=32 time=61ms TTL=56
Reply from 192.168.116.171: bytes=32 time=32ms TTL=56
Reply from 192.168.116.171: bytes=32 time=34ms TTL=56
Ping statistics for 192.168.116.171:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 31ms, Maximum = 61ms, Average = 39ms
```

2. Multiply that value by the network bandwidth. For example, if average RTT is .08 seconds, and the bandwidth is 100 megabits per second, then the optimum buffer size is:

$0.08 \text{ second} * 100 \text{ megabits per second} = 8 \text{ megabits}$

3. Divide the result by 8 to determine the number of bytes (8 bits to a byte). For example:

$8 \text{ megabits} / 8 = 1 \text{ megabyte per second}$

The required unit for `TCPBUFSIZE` is bytes, so you would set it to a value of 1000000.

The maximum socket buffer size for non-Windows systems is usually limited by default. Ask your system administrator to increase the default value on the source and target systems so that Oracle GoldenGate can increase the buffer size configured with `TCPBUFSIZE`.

Eliminating Disk I/O Bottlenecks

I/O activity can cause bottlenecks for both Extract and Replicat.

- A regular Extract generates disk writes to a trail and disk reads from a data source.
- A data pump and Replicat generate disk reads from a local trail.
- Each process writes a recovery checkpoint to its checkpoint file on a regular schedule.
- [Improving I/O performance Within the System Configuration](#)
- [Improving I/O Performance Within the Oracle GoldenGate Configuration](#)

Improving I/O performance Within the System Configuration

If there are I/O waits on the disk subsystems that contain the trail files, put the trails on the fastest disk controller possible.

Check the RAID configuration. Because Oracle GoldenGate writes data sequentially, RAID 0+1 (striping and mirroring) is a better choice than RAID 5, which uses checksums that slow down I/O and are not necessary for these types of files.

Improving I/O Performance Within the Oracle GoldenGate Configuration

You can improve I/O performance by making configurations changes within Oracle GoldenGate. Try increasing the values of the following parameters.

- Use the `CHECKPOINTSECS` parameter to control how often Extract and Replicat make their routine checkpoints.

 **Note:**

`CHECKPOINTSECS` is not valid for an integrated Replicat on an Oracle database system.

- Use the `GROUPTRANSOPS` parameter to control the number of SQL operations that are contained in a Replicat transaction when operating in its normal mode. Increasing the number of operations in a Replicat transaction improves the performance of Oracle GoldenGate by reducing the number of transactions executed by Replicat, and by reducing I/O activity to the checkpoint file and the checkpoint table, if used. Replicat issues a checkpoint whenever it applies a transaction to the target, in addition to its scheduled checkpoints.

 **Note:**

`GROUPTRANSOPS` is not valid for an integrated Replicat on an Oracle database system, unless the inbound server parameter `parallelism` is set to 1.

- Use the `EOFDELAY` or `EOFDELAYCSECS` parameter to control how often Extract, a data pump, or Replicat checks for new data after it has reached the end of the current data in its data source. You can reduce the system I/O overhead of these reads by increasing the value of this parameter.

 **Note:**

Increasing the values of these parameters improves performance, but it also increases the amount of data that must be reprocessed if the process fails. This has an effect on overall latency between source and target. Some testing will help you determine the optimal balance between recovery and performance.

Managing Virtual Memory and Paging

Because Oracle GoldenGate replicates only committed transactions, it stores the operations of each transaction in a managed virtual-memory pool known as a *cache* until it receives either a commit or a rollback for that transaction. One global cache

operates as a shared resource of an Extract or Replicat process. The Oracle GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that Oracle GoldenGate processes work in a sustained and efficient manner. The `CACHEMGR` parameter controls the amount of virtual memory and temporary disk space that is available for caching uncommitted transaction data that is being processed by Oracle GoldenGate.

When a process starts, the cache manager checks the availability of resources for virtual memory, as shown in the following example:

```
CACHEMGR virtual memory values (may have been adjusted)CACHESIZE:  
32GCACHEPAGEOUTSIZE (normal): 8M PROCESS VM AVAIL FROM OS (min): 63.97GCACHESIZEMAX  
(strict force to disk): 48G
```

If the current resources are not sufficient, a message like the following may be returned:

```
2013-11-11 14:16:22 WARNING OGG-01842 CACHESIZE PER DYNAMIC DETERMINATION (32G) LESS  
THAN RECOMMENDED: 64G (64bit system)vm found: 63.97GCheck swap space. Recommended  
swap/extract: 128G (64bit system).
```

If the system exhibits excessive paging and the performance of critical processes is affected, you can reduce the `CACHESIZE` option of the `CACHEMGR` parameter. You can also control the maximum amount of disk space that can be allocated to the swap directory with the `CACHEDIRECTORY` option. For more information about `CACHEMGR`, see *Reference for Oracle GoldenGate*.

Optimizing Data Filtering and Conversion

Heavy amounts of data filtering or data conversion add processing overhead. The following are suggestions for minimizing the impact of this overhead on the other processes on the system.

- Avoid using the primary Extract to filter and convert data. Keep it dedicated to data capture. It will perform better and is less vulnerable to any process failures that result from those activities. The objective is to make certain the primary Extract process is running and keeping pace with the transaction volume.
- Use Replicat or a data-pump to perform filtering and conversion. Consider any of the following configurations:
 - Use a data pump on the source if the system can tolerate the overhead. This configuration works well when there is a high volume of data to be filtered, because it uses less network bandwidth. Only filtered data gets sent to the target, which also can help with security considerations.
 - Use a data pump on an intermediate system. This configuration keeps the source and target systems free of the overhead, but uses more network bandwidth because unfiltered data is sent from the source to the intermediate system.
 - Use a data pump or Replicat on the target if the system can tolerate the overhead, and if there is adequate network bandwidth for sending large amounts of unfiltered data.
- If you have limited system resources, a least-best option is to divide the filtering and conversion work between Extract and Replicat.

Tuning Replicat Transactions

Replicat uses regular SQL, so its performance depends on the performance of the target database and the type of SQL that is being applied (inserts, versus updates or deletes). However, you can take certain steps to maximize Replicat efficiency.

Topics:

- [Tuning Coordination Performance Against Barrier Transactions](#)
- [Applying Similar SQL Statements in Arrays](#)
- [Preventing Full Table Scans in the Absence of Keys](#)
- [Splitting Large Transactions](#)
- [Adjusting Open Cursors](#)
- [Improving Update Speed](#)
- [Set a Replicat Transaction Timeout](#)

Tuning Coordination Performance Against Barrier Transactions

In a coordinated Replicat configuration, barrier transactions such as updates to the primary key cause an increased number of commits to the database, and they interrupt the benefit of the `GROUPTRANSOPS` feature of Replicat. When there is a high number of barrier transactions in the overall workload of the coordinated Replicat, using a high number of threads can actually degrade Replicat performance.

To maintain high performance when large numbers of barrier transactions are expected, you can do the following:

- Reduce the number of active threads in the group. This reduces the overall number of commits that Replicat performs.
- Move the tables that account for the majority of the barrier transactions, and any tables with which they have dependencies, to a separate coordinated Replicat group that has a small number of threads. Keep the tables that have minimal barrier transactions in the original Replicat group with the higher number of threads, so that parallel performance is maintained without interruption by barrier transactions.
- (Oracle RAC) In a new Replicat configuration, you can increase the `PCTFREE` attribute of the Replicat checkpoint table. However, this must be done before Replicat is started for the first time. The recommended value of `PCTFREE` is 90.

Applying Similar SQL Statements in Arrays

Use the `BATCHSQL` parameter to increase the performance of Replicat. `BATCHSQL` causes Replicat to organize similar SQL statements into arrays and apply them at an accelerated rate. In its normal mode, Replicat applies one SQL statement at a time.

When Replicat is in `BATCHSQL` mode, smaller row changes will show a higher gain in performance than larger row changes. At 100 bytes of data per row change, `BATCHSQL` has been known to improve the performance of Replicat by up to 300 percent, but actual performance benefits will vary, depending on the mix of operations. At around 5,000 bytes of data per row change, the benefits of using `BATCHSQL` diminish.

The gathering of SQL statements into batches improves efficiency but also consumes memory. To maintain optimum performance, use the following `BATCHSQL` options:

```
BATCHESPERQUEUE  
BYTESPERQUEUE  
OPSPERBATCH  
OPSPERQUEUE
```

As a benchmark for setting values, assume that a batch of 1,000 SQL statements at 500 bytes each would require less than 10 megabytes of memory.

You can use `BATCHSQL` with the `BATCHTRANSOPS` option to tune array sizing. `BATCHTRANSOPS` controls the maximum number of batch operations that can be grouped into a transaction before requiring a commit. The default for non-integrated Replicat is 1000. The default for integrated Replicat is 50. If there are many wait dependencies when using integrated Replicat, try reducing the value of `BATCHTRANSOPS`. To determine the number of wait dependencies, view the `TOTAL_WAIT_DEPS` column of the `V$GG_APPLY_COORDINATOR` database view in the Oracle database.

See *Reference for Oracle GoldenGate* for additional usage considerations and syntax.

Preventing Full Table Scans in the Absence of Keys

If a target table does not have a primary key, a unique key, or a unique index, Replicat uses all of the columns to build its `WHERE` clause. This is, essentially, a full table scan.

To make row selection more efficient, use a `KEYCOLS` clause in the `TABLE` and `MAP` statements to identify one or more columns as unique. Replicat will use the specified columns as a key. The following example shows a `KEYCOLS` clause in a `TABLE` statement:

```
TABLE hr.emp, KEYCOLS (FIRST_NAME, LAST_NAME, DOB, ID_NO);
```

For usage guidelines and syntax, see the `TABLE` and `MAP` parameters in *Reference for Oracle GoldenGate*.

Splitting Large Transactions

If the target database cannot handle large transactions from the source database, you can split them into a series of smaller ones by using the Replicat parameter `MAXTRANSOPS`. See *Reference for Oracle GoldenGate* for more information.



Note:

`MAXTRANSOPS` is not valid for an integrated Replicat on an Oracle database system.

Adjusting Open Cursors

The Replicat process maintains cursors for cached SQL statements and for `SQLEXEC` operations. Without enough cursors, Replicat must age more statements. By default, Replicat maintains as many cursors as allowed by the `MAXSQLSTATEMENTS` parameter. You might find that the value of this parameter needs to be increased. If so, you might

also need to adjust the maximum number of open cursors that are permitted by the database. See *Reference for Oracle GoldenGate* for more information.

Improving Update Speed

Excessive block fragmentation causes Replicat to apply SQL statements at a slower than normal speed. Reorganize heavily fragmented tables, and then stop and start Replicat to register the new object ID.

Set a Replicat Transaction Timeout

Use the `TRANSACTIONTIMEOUT` parameter to prevent an uncommitted Replicat target transaction from holding locks on the target database and consuming its resources unnecessarily. You can change the value of this parameter so that Replicat can work within existing application timeouts and other database requirements on the target.

`TRANSACTIONTIMEOUT` limits the amount of time that Replicat can hold a target transaction open if it has not received the end-of-transaction record for the last source transaction in that transaction. By default, Replicat groups multiple source transactions into one target transaction to improve performance, but it will not commit a partial source transaction and will wait indefinitely for that last record. The Replicat parameter `GROUPTRANSOPS` controls the minimum size of a grouped target transaction.

The following events could last long enough to trigger `TRANSACTIONTIMEOUT`:

- Network problems prevent trail data from being delivered to the target system.
- Running out of disk space on any system, preventing trail data from being written.
- Collector abends (a rare event).
- Extract abends or is terminated in the middle of writing records for a transaction.
- An Extract data pump abends or is terminated.
- There is a source system failure, such as a power outage or system crash.

See *Reference for Oracle GoldenGate* for more information.

19

Performing Administrative Operations

This chapter contains instructions for making changes to applications, systems, and Oracle GoldenGate while the replication environment is active and processing data changes.

Topics:

- [Performing Application Patches](#)
- [Initializing the Transaction Logs](#)
- [Shutting Down the System](#)
- [Changing Database Attributes](#)
- [Adding Process Groups to an Active Configuration](#)
- [Changing the Size of Trail Files](#)
- [Switching Extract from Classic Mode to Integrated Mode](#)
- [Switching Extract from Integrated Mode to Classic Mode](#)
- [Switching Replicat from Nonintegrated Mode to Integrated Mode](#)
- [Switching Replicat from Integrated Mode to Nonintegrated Mode](#)
- [Switching Replicat to Coordinated Mode](#)
- [Administering a Coordinated Replicat Configuration](#)
- [Restarting a Primary Extract after System Failure or Corruption](#)

Performing Application Patches

Application patches and application upgrades typically perform DDL such as adding new objects or changing existing objects. To apply applications patches or upgrades in an Oracle GoldenGate environment, you can do one of the following:

- If Oracle GoldenGate supports DDL replication for your database type, you can use it to replicate the DDL without stopping replication processes. To use this method, the source and target table structures must be identical.
- You can apply the patch or upgrade manually on both source and target after taking the appropriate steps to ensure replication continuity.

To Use Oracle GoldenGate to Replicate Patch DDL

1. If you have not already done so, dedicate some time to learn, install, and configure the Oracle GoldenGate DDL support. See the instructions for your database in this documentation. Once the DDL environment is in place, future patches and upgrades will be easier to apply.
2. If the application patch or upgrade adds new objects that you want to include in data replication, make certain that you include them in the DDL parameter statement. To add new objects to your `TABLE` and `MAP` statements, see the procedure on [Adding Tables to the Oracle GoldenGate Configuration](#).

3. If the application patch or upgrade installs triggers or cascade constraints, disable those objects on the target to prevent collisions between DML that they execute on the target and the same DDL that is replicated from the source trigger or cascaded operation.

To Apply a Patch Manually on the Source and Target

1. Stop access to the source database.
2. Allow Extract to finish capturing the transaction data that remains in the transaction log. To determine when Extract is finished, issue the following command in GGSCI until it returns `At EOF`.

```
SEND EXTRACT group GETLAG
```

3. Stop Extract.

```
STOP EXTRACT group
```

4. Start applying the patch on the source.
5. Wait until the data pump (if used) and Replicat are finished processing the data in their respective trails. To determine when they are finished, use the following commands until they return `At EOF`.

```
SEND EXTRACT group GETLAG  
SEND REPLICAT group GETLAG
```

6. Stop the data pump and Replicat.

```
STOP EXTRACT group  
STOP REPLICAT group
```

At this point, the data in the source and target should be identical, because all of the replicated transactional changes from the source have been applied to the target.

7. Apply the patch on the target.
8. If the patches changed table definitions, run DEFGEN for the source tables to generate updated source definitions, and then replace the old definitions with the new ones in the existing source definitions file on the target system.
9. Start the Oracle GoldenGate processes whenever you are ready to begin capturing user activity again.

Initializing the Transaction Logs

When you initialize a transaction log, you must ensure that all of the data is processed by Oracle GoldenGate first, and then you must delete and re-add the Extract group and its associated trail.

1. Stop the application from accessing the database. This stops more transaction data from being logged.
2. Run GGSCI and issue the `SEND EXTRACT` command with the `LOGEND` option for the primary Extract group. This command queries Extract to determine whether or not Extract is finished processing the records that remain in the transaction log.

```
SEND EXTRACT group LOGEND
```


3. Continue issuing the command until it returns a YES status, indicating that there are no more records to process.
4. On the target system, run GGSCI and issue the `SEND REPLICAT` command with the `STATUS` option. This command queries Replicat to determine whether or not it is finished processing the data that remains in the trail.

```
SEND REPLICAT group STATUS
```

5. Continue issuing the command until it shows 0 records in the current transaction, for example:

```
Sending STATUS request to REPLICAT REPSTAB...  
Current status:  
  Seqno 0, Rba 9035  
  0 records in current transaction.
```

6. Stop the primary Extract group, the data pump (if used), and the Replicat group.

```
STOP EXTRACT group  
STOP EXTRACT pump_group  
STOP REPLICAT group
```

7. Delete the Extract, data pump, and Replicat groups.

```
DELETE EXTRACT group  
DELETE EXTRACT pump_group  
DELETE REPLICAT group
```

8. Using standard operating system commands, delete the trail files.

9. Stop the database.

10. Initialize and restart the database.

11. Recreate the primary Extract group.

```
ADD EXTRACT group TRANLOG, BEGIN NOW
```

12. Recreate the local trail (if used).

```
ADD EXTTRAIL trail, EXTRACT group
```

13. Recreate the data pump (if used).

```
ADD EXTRACT pump_group, EXTTRAILSOURCE trail
```

14. Recreate the remote trail.

```
ADD RMTTRAIL trail, EXTRACT pump_group
```

15. Recreate the Replicat group.

```
ADD REPLICAT group, EXTTRAIL trail
```

16. Start Extract, the data pump (if used), and Replicat.

```
START EXTRACT group  
START EXTRACT pump_group  
START REPLICAT group
```

Shutting Down the System

When shutting down a system for maintenance and other procedures that affect Oracle GoldenGate, follow these steps to make certain that Extract has processed all of the transaction log records. Otherwise, you might lose synchronization data.

1. Stop all application and database activity that generates transactions that are processed by Oracle GoldenGate.
2. Run GGSCI.
3. In GGSCI, issue the `SEND EXTRACT` command with the `LOGEND` option. This command queries the Extract process to determine whether or not it is finished processing the records in the data source.

```
SEND EXTRACT group LOGEND
```

4. Continue issuing the command until it returns a `YES` status. At that point, all transaction log data has been processed, and you can safely shut down Oracle GoldenGate and the system.

Changing Database Attributes

This section addresses administrative operations that are performed on database tables and structures.

Topics:

- [Changing Database Metadata](#)
- [Adding Tables to the Oracle GoldenGate Configuration](#)
- [Coordinating Table Attributes between Source and Target](#)
- [Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables](#)
- [Dropping and Recreating a Source Table](#)
- [Changing the Number of Oracle RAC Threads when Using Classic Capture](#)
- [Changing the ORACLE_SID](#)
- [Purging Archive Logs](#)
- [Reorganizing a DB2 Table \(z/OS Platform\)](#)

Changing Database Metadata

This procedure is required to prevent Replicat errors when changing the following metadata of the source database:

- Database character set
- National character set
- Locale
- Timezone
- Object name case-sensitivity

If these changes are made without performing this procedure, the following error occurs:

```
2013-05-26 20:10:09 ERROR OGG-05500 Detected database metadata mismatch between
current trail file ./dirdat/_p/v1000000003 and the previous sequence. *DBTIMEZONE:
[GMT]/[UTC].
```

This procedure stops Extract, and then creates a new trail file. The new database metadata is included in this new file with the transactions that started after the change.

1. Stop transaction activity on the source database. Do not make the metadata change to the database yet.
2. In GGSCI on the source system, issue the `SEND EXTRACT` command with the `LOGEND` option until it shows there is no more redo data to capture.

```
SEND EXTRACT group LOGEND
```

3. Stop Extract.

```
STOP EXTRACT group
```

4. On each target system, issue the `SEND REPLICAT` command with the `STATUS` option until it shows a status of "At EOF" to indicate that it finished processing all of the data in the trail. This must be done on all target systems until all Replicat processes return "At EOF."

```
SEND REPLICAT group STATUS
```

5. Stop the data pumps and Replicat.

```
STOP EXTRACT group
STOP REPLICAT group
```

6. Change the database metadata.

7. In GGSCI on the source system, issue the `ALTER EXTRACT` command with the `ETROLLOVER` option for the primary Extract to roll over the local trail to the start of a new file.

```
ALTER EXTRACT group, ETROLLOVER
```

8. Issue the `ALTER EXTRACT` command with the `ETROLLOVER` option for the data pumps to roll over the remote trail to the start of a new file.

```
ALTER EXTRACT pump, ETROLLOVER
```

9. Start Extract.

```
START EXTRACT group
```

10. In GGSCI, reposition the data pumps and Replicat processes to start at the new trail sequence number.

```
ALTER EXTRACT pump, EXTSEQNO seqno, EXTRBA RBA
ALTER REPLICAT group, EXTSEQNO seqno, EXTRBA RBA
```

11. Start the data pumps.

```
START EXTRACT group
```

12. Start the Replicat processes.

```
START REPLICAT group
```

Adding Tables to the Oracle GoldenGate Configuration

This procedure assumes that the Oracle GoldenGate DDL support feature is not in use, or is not supported for, your database.

Note:

For Oracle and MySQL databases, you can enable the DDL support feature of Oracle GoldenGate to automatically capture and apply the DDL that adds new tables, instead of using this procedure. See the appropriate instructions for your database in this documentation.

Review these steps before starting. The process varies slightly, depending on whether or not the new tables satisfy wildcards in the `TABLE` parameter, and whether or not names or data definitions must be mapped on the target.

Prerequisites for Adding Tables to the Oracle GoldenGate Configuration

- This procedure assumes that the source and target tables are either empty or contain identical (already synchronized) data.
- You may be using the `DBLOGIN` and `ADD TRANDATA` commands. Before starting this procedure, see *Reference for Oracle GoldenGate* for the proper usage of these commands for your database.

To Add a Table to the Oracle GoldenGate Configuration

1. Stop user access to the new tables.
2. *(If new tables do not satisfy a wildcard)* If you are adding numerous tables that do not satisfy a wildcard, make a copy of the Extract and Replicat parameter files, and then add the new tables with `TABLE` and `MAP` statements. If you do not want to work with a copy, then edit the original parameter files after you are prompted to stop each process.
3. *(If new tables satisfy wildcards)* In the Extract and Replicat parameter files, make certain the `WILDCARDRESOLVE` parameter is not being used, unless it is set to the default of `DYNAMIC`.
4. *(If new tables do not satisfy a wildcard)* If the new tables *do not* satisfy a wildcard definition, stop Extract.

```
STOP EXTRACT group
```

5. Add the new tables to the source and target databases.
6. If required for the source database, issue the `ADD TRANDATA` command in GGSCI for the new tables. Before using `ADD TRANDATA`, issue the `DBLOGIN` command.
7. Depending on whether the source and target definitions are identical or different, use either `ASSUMETARGETDEFS` or `SOURCEDEFS` in the Replicat parameter file. If `SOURCEDEFS` is needed, you can do either of the following:
 - Run `DEFGEN`, then copy the new definitions to the source definitions file on the target.

- If the new tables match a definitions template, specify the template with the `DEF` option of the `MAP` parameter. (DEFGEN not needed.)
8. To register the new source definitions or new `MAP` statements, stop and then start Replicat.

```
STOP REPLICAT group
START REPLICAT group
```

9. Start Extract, if applicable.

```
START EXTRACT group
```

10. Permit user access to the new tables.

Coordinating Table Attributes between Source and Target

Follow this procedure if you are changing an attribute of a source table that is in the Oracle GoldenGate configuration, such as adding or changing columns or partitions, or changing supplemental logging details (Oracle). It directs you how to make the same change to the target table without incurring replication latency.

Note:

See also [Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables](#).

Note:

This procedure assumes that the Oracle GoldenGate DDL support feature is not in use, or is not supported for your database. For Oracle and MySQL databases, you can enable the DDL support feature of Oracle GoldenGate to propagate the DDL changes to the target, instead of using this procedure.

1. On the source and target systems, create a table, to be known as the *marker table*, that can be used for the purpose of generating a marker that denotes a stopping point in the transaction log. Just create two simple columns: one as a primary key and the other as a regular column. For example:

```
CREATE TABLE marker
(
  id int NOT NULL,
  column varchar(25) NOT NULL,
  PRIMARY KEY (id)
);
```

2. Insert a row into the marker table on both the source and target systems.

```
INSERT INTO marker VALUES (1, 1);
COMMIT;
```

3. On the source system, run GGSCI.
4. Open the Extract parameter file for editing.

▲ Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

5. Add the marker table to the Extract parameter file in a `TABLE` statement.

```
TABLE marker;
```

6. Save and close the parameter file.
7. Add the marker table to the `TABLE` statement of the data pump, if one is being used.
8. Stop the Extract and data pump processes, and then restart them immediately to prevent capture lag.

```
STOP EXTRACT group
START EXTRACT group
STOP EXTRACT pump_group
START EXTRACT pump_group
```

9. On the target system, run GGSCI.
10. Open the Replicat parameter file for editing.

▲ Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

11. Add the marker table to the Replicat parameter file in a `MAP` statement, and use the `EVENTACTIONS` parameter as shown to stop Replicat and ignore operations on the marker table.

```
MAP marker, TARGET marker, EVENTACTIONS (STOP, IGNORE);
```

12. Save and close the parameter file.
13. Stop, and then immediately restart, the Replicat process.

```
STOP REPLICAT group
START REPLICAT group
```

14. When you are ready to change the table attributes for both source and target tables, stop all user activity on them.
15. On the source system, perform an `UPDATE` operation to the marker table as the only operation in the transaction.

```
UPDATE marker  
SET column=2,  
WHERE id=1;  
COMMIT;
```

16. On the target system, issue the following command until it shows that Replicat is stopped as a result of the `EVENTACTIONS` rule.

```
STATUS REPLICAT group
```

17. Perform the DDL on the source and target tables, but do not yet allow user activity.
18. Start Replicat.

```
START REPLICAT group
```

19. Allow user activity on the source and target tables.

Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables

To add a fixed length column to a table that is in reordered row format and contains one or more variable length columns, one of the following will be required, depending on whether the table can be quiesced or not.

If the Table can be Quiesced

1. Allow Extract to finish capturing transactions that happened prior to the quiesce.
2. Alter the table to add the column.
3. Reorganize the tablespace.
4. Restart Extract.
5. Allow table activity to resume.

If the Table cannot be Quiesced

1. Stop Extract.
2. Remove the table from the `TABLE` statement in the parameter file.
3. Restart Extract.
4. Alter the table to add the column.
5. Reorganize the tablespace.
6. Stop Extract.
7. Add the table back to the `TABLE` statement.
8. Resynchronize the source and target tables.
9. Start Extract.
10. Allow table activity to resume.

Dropping and Recreating a Source Table

Dropping and recreating a source table requires caution when performed while Oracle GoldenGate is running.

1. Stop access to the table.

2. Allow Extract to process any remaining changes to that table from the transaction logs. To determine when Extract is finished, use the `INFO EXTRACT` command in GGSCI.

```
INFO EXTRACT group
```

3. Stop Extract.

```
STOP EXTRACT group
```

4. Drop and recreate the table.
5. If supported for this database, run the `ADD TRANDATA` command in GGSCI for the table.
6. If the recreate action changed the source table's definitions so that they are different from those of the target, run the `DEFGEN` utility for the source table to generate source definitions, and then replace the old definitions with the new definitions in the *existing* source definitions file on the target system.
7. Permit user access to the table.

Changing the Number of Oracle RAC Threads when Using Classic Capture

Valid for Extract in classic capture mode for Oracle. When Extract operates in classic capture mode, the Extract group must be dropped and re-added any time the number of redo threads in an Oracle RAC cluster changes. To drop and add an Extract group, perform the following steps:

1. On the source and target systems, run GGSCI.
2. Stop Extract and Replicat.

```
STOP EXTRACT group  
STOP REPLICAT group
```

3. On the source system, issue the following command to delete the primary Extract group and the data pump.

```
DELETE EXTRACT group  
DELETE EXTRACT pump_group
```

4. On the target system, issue the following command to delete the Replicat groups.

```
DELETE REPLICAT group
```

5. Using standard operating system commands, remove the local and remote trail files.
6. Add the primary Extract group again with the same name as before, specifying the new number of RAC threads.

```
ADD EXTRACT group TRANLOG, THREADS n, BEGIN NOW
```

7. Add the local trail again with the same name as before.

```
ADD EXTTRAIL trail, EXTRACT group
```

8. Add the data pump Extract again, with the same name as before.

```
ADD EXTRACT group EXTTRAILSOURCE trail, BEGIN NOW
```

9. Add the remote trail again with the same name as before.


```
ADD RMTTRAIL trail, EXTRACT group
```

10. Add the Replicat group with the same name as before. Leave off any `BEGIN` options so that processing begins at the start of the trail.

```
ADD REPLICAT group EXTTRAIL trail
```

11. Start all processes, using wildcards as appropriate. If the re-created processes are the only ones in the source and target Oracle GoldenGate instances, you can use `START ER *` instead of the following commands.

```
START EXTRACT group  
START REPLICAT group
```

Changing the ORACLE_SID

You can change the `ORACLE_SID` and `ORACLE_HOME` without having to change environment variables at the operating-system level. Depending on whether the change is for the source or target database, set the following parameters in the Extract or Replicat parameter files. Then, stop and restart Extract or Replicat for the parameters to take effect.

```
SETENV (ORACLE_HOME=location)  
SETENV (ORACLE_SID='SID')
```

Purging Archive Logs

An Oracle archive log can be purged safely once Extract's read and write checkpoints are past the end of that log. Extract does not write a transaction to a trail until it has been committed, so Extract must keep track of all open transactions. To do so, Extract requires access to the archive log where each open transaction started and all archive logs thereafter.

Extract reads the current archive log (the read checkpoint) for new transactions and also has a checkpoint (the recovery checkpoint) in the oldest archive log for which there is an uncommitted transaction.

Use the following command in GGSCI to determine Extract's checkpoint positions.

```
INFO EXTRACT group, SHOWCH
```

- The `Input Checkpoint` field shows where Extract began processing when it was started.
- The `Recovery Checkpoint` field shows the location of the oldest uncommitted transaction.
- The `Next Checkpoint` field shows the position in the redo log that Extract is reading.
- The `Output Checkpoint` field shows the position where Extract is writing.

You can write a shell script that purges all archive logs no longer needed by Extract by capturing the sequence number listed under the `Recovery Checkpoint` field. All archive logs prior to that one can be safely deleted.

Reorganizing a DB2 Table (z/OS Platform)

When using IBM's REORG utility to reorganize a DB2 table that has compressed tablespaces, specify the `KEEPDICTIONARY` option if the table is being processed by Oracle GoldenGate. This prevents the REORG utility from recreating the compression dictionary, which would cause log data that was written prior to the change not to be decompressed and cause Extract to terminate abnormally. As an alternative, ensure that all of the changes for the table have been extracted by Oracle GoldenGate before doing the reorganization, or else truncate the table.

Adding Process Groups to an Active Configuration

This section describes how to add process groups.

Topics:

- [Before You Start](#)
- [Adding Another Extract Group to an Active Configuration](#)
- [Adding Another Data Pump to an Active Configuration](#)
- [Adding Another Replicat Group to an Active Configuration](#)

Before You Start

These instructions are for adding process groups to a configuration that is already active. The procedures should be performed by someone who has experience with Oracle GoldenGate. They involve stopping processes for a short period of time and reconfiguring parameter files. The person performing them must:

- Know the basic components of an Oracle GoldenGate configuration
- Understand Oracle GoldenGate parameters and commands
- Have access to GGSCI to create groups and parameter files
- Know which parameters to use in specific situations

Instructions are provided for:

- [Adding Another Extract Group to an Active Configuration](#)
- [Adding Another Data Pump to an Active Configuration](#)
- [Adding Another Replicat Group to an Active Configuration](#)

Adding Another Extract Group to an Active Configuration

This procedure splits the workload of an existing Extract group into multiple Extract groups. It also provides instructions for including a data pump group (if applicable) and a Replicat group to propagate data that is captured by the new Extract group.

Steps are performed on the source and target systems.

1. Make certain the archived transaction logs are available in case the online logs recycle before you complete this procedure.
2. Choose a name for the new Extract group.

3. Decide whether or not to use a data pump.
4. On the source system, run GGSCI.
5. Create a parameter file for the new Extract group.

```
EDIT PARAMS group
```

 **Note:**

You can copy the original parameter file to use for this group, but make certain to change the Extract group name and any other relevant parameters that apply to this new group.

6. In the parameter file, include:
 - EXTRACT parameter that specifies the new group.
 - Appropriate database login parameters.
 - Other appropriate Extract parameters for your configuration.
 - EXTTRAIL parameter that points to a local trail (if you will be adding a data pump) or a RMTTRAIL parameter (if you are not adding a data pump).
 - RMTHOST parameter if this Extract will write directly to a remote trail.
 - TABLE statement(s) (and TABLEEXCLUDE, if appropriate) for the tables that are to be processed by the new group.
7. Save and close the file.
8. Edit the original Extract parameter file(s) to remove the TABLE statements for the tables that are being moved to the new group or, if using wildcards, add the TABLEEXCLUDE parameter to exclude them from the wildcard specification.
9. (Oracle) If you are using Extract in integrated mode, register the new Extract group with the source database.

```
REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])]
```

10. Lock the tables that were moved to the new group, and record the timestamp for the point when the locks were applied. For Oracle tables, you can run the following script, which also releases the lock after it is finished.

```
-- temp_lock.sql
-- use this script to temporary lock a table in order to
-- get a timestamp

lock table &schema . &table_name in EXCLUSIVE mode;
SELECT TO_CHAR(sysdate, 'MM/DD/YYYY HH24:MI:SS') "Date" FROM dual;
commit;
```

11. Unlock the table(s) if you did not use the script in the previous step.
12. Stop the old Extract group(s) and any existing data pumps.

```
STOP EXTRACT group
```

13. Add the new Extract group and configure it to start at the timestamp that you recorded.

```
ADD EXTRACT group, TRANLOG, BEGIN YYYY/MM/DD HH:MI:SS:CCCCCC
```

14. Add a trail for the new Extract group.

```
ADD {EXTTRAIL | RMTTRAIL} trail, EXTRACT group
```

Where:

- **EXTTRAIL** creates a local trail. Use this option if you will be creating a data pump for use with the new Extract group. Specify the trail that is specified with **EXTTRAIL** in the parameter file. After creating the trail, go [To Link a Local Data Pump to the New Extract Group](#).
- **RMTTRAIL** creates a remote trail. Use this option if a data pump will not be used. Specify the trail that is specified with **RMTTRAIL** in the parameter file. After creating the trail, go [To Link a Remote Replicat to the New Data Pump](#).

You can specify a relative or full path name. Examples:

```
ADD RMTTRAIL dirdat/rt, EXTRACT primary
ADD EXTTRAIL c:\ogg\dirdat\lt, EXTRACT primary
```

To Link a Local Data Pump to the New Extract Group

1. On the source system, add the data-pump Extract group using the **EXTTRAIL** trail as the data source.

```
ADD EXTRACT pump, EXTTRAILSOURCE trail
```

For example:

```
ADD EXTRACT pump2, EXTTRAILSOURCE dirdat\lt
```

2. Create a parameter file for the data pump.

```
EDIT PARAMS pump
```

3. In the parameter file, include the appropriate Extract parameters for your configuration, plus:

- **RMTHOST** parameter to point to the target system.
- **RMTTRAIL** parameter to point to a new remote trail (to be specified later).
- **TABLE** parameter(s) for the tables that are to be processed by this data pump.

4. In GGSCI on the source system, add a remote trail for the data-pump. Use the trail name that you specified with **RMTTRAIL** in the parameter file.

```
ADD RMTTRAIL trail, EXTRACT pump
```

For example:

```
ADD RMTTRAIL dirdat/rt, EXTRACT pump2
```

5. Follow the steps in [To Link a Remote Replicat to the New Data Pump](#).

To Link a Remote Replicat to the New Data Pump

1. In GGSCI on the target system, add a Replicat group to read the remote trail. For **EXTTRAIL**, specify the same trail as in the **RMTTRAIL** Extract parameter and the **ADD RMTTRAIL** command.

```
ADD REPLICAT group, EXTTRAIL trail
```

For example:

```
ADD REPLICAT rep2, EXTTRAIL /home/ggs/dirdat/rt
```

2. Create a parameter file for this Replicat group. Use `MAP` statement(s) to specify the same tables that you specified for the new primary Extract and the data pump (if used).
3. On the source system, start the Extract groups and data pumps.

```
START EXTRACT group
START EXTRACT pump
```

4. On the target system, start the new Replicat group.

```
START REPLICAT group
```

Adding Another Data Pump to an Active Configuration

This procedure adds a data-pump Extract group to an active primary Extract group on the source system. It makes these changes:

- The primary Extract will write to a local trail.
- The data pump will write to a new remote trail after the data in the old trail is applied to the target.
- The old Replicat group will be replaced by a new one.

Steps are performed on the source and target systems.

1. On the source system, run GGSCI.
2. Add a local trail, using the name of the primary Extract group for *group*.

```
ADD EXTTRAIL trail, EXTRACT group
```

For example:

```
ADD EXTTRAIL dirdat\lt, EXTRACT primary
```

3. Open the parameter file of the primary Extract group, and replace the `RMTTRAIL` parameter with an `EXTTRAIL` parameter that points to the local trail that you created.

Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

Example `EXTTRAIL` parameter:

```
EXTTRAIL dirdat\lt
```

4. Remove the `RMTHOST` parameter.
5. Save and close the file.

6. Add a new data-pump Extract group, using the trail that you specified in step 2 as the data source.

```
ADD EXTRACT group, EXTTRAILSOURCE trail
```

For example:

```
ADD EXTRACT pump, EXTTRAILSOURCE dirdat\lt
```

7. Create a parameter file for the new data pump.

```
EDIT PARAMS group
```

8. In the parameter file, include the appropriate Extract parameters for your configuration, plus:
 - TABLE parameter(s) for the tables that are to be processed by this data pump.
 - RMTHOST parameter to point to the target system.
 - RMTTRAIL parameter to point to a new remote trail (to be created later).
9. In GGSCI on the source system, add a remote trail for the data-pump. Use the trail name that is specified with RMTTRAIL in the data pump's parameter file, and specify the group name of the data pump for EXTRACT.

```
ADD RMTTRAIL trail, EXTRACT group
```

For example:

```
ADD RMTTRAIL dirdat/rt, EXTRACT pump
```

 **Note:**

This command binds a trail name to an Extract group but does not actually create the trail. A trail file is created when processing starts.

10. On the target system, run GGSCI.
11. Add a new Replicat group and link it with the remote trail.

```
ADD REPLICAT group, EXTTRAIL trail
```

For example:

```
ADD REPLICAT rep, EXTTRAIL dirdat/rt
```

12. Create a parameter file for this Replicat group. You can copy the parameter file from the original Replicat group, but make certain to change the REPLICAT parameter to the new group name.
13. On the source system, stop the primary Extract group, then start it again so that the parameter changes you made take effect.

```
STOP EXTRACT group  
START EXTRACT group
```

14. On the source system, start the data pump.

```
START EXTRACT group
```

15. On the target system, issue the LAG REPLICAT command for the old Replicat, and continue issuing it until it reports At EOF, no more records to process.

```
LAG REPLICAT group
```

16. Stop the old Replicat group.

```
STOP REPLICAT group
```

17. If using a checkpoint table for the old Replicat group, log into the database from GGSCI.

```
DBLOGIN [SOURCEDB datasource] [{, USERIDALIAS alias | USERID user [,options]]
```

18. Delete the old Replicat group.

```
DELETE REPLICAT group
```

19. Start the new Replicat group.

```
START REPLICAT group
```

 **Note:**

Do not delete the old remote trail, just in case it is needed later on for a support case or some other reason. You can move it to another location, if desired.

Adding Another Replicat Group to an Active Configuration

This procedure adds a new Replicat group to an existing Replicat group. The new Replicat reads from the same trail as the original Replicat.

Multiple Replicat groups may be required when Replicat is configured in classic mode, for the purpose of isolating transactions on certain tables or improving performance. Multiple Replicat groups usually are not required if using coordinated Replicat, because you can divide the workload among multiple processing threads within the same Replicat group. See [Creating an Online Replicat Group](#) for more information about Replicat modes.

Steps are performed on the source and target systems.

1. Choose a name for the new group.
2. On the target system, run GGSCI.
3. Create a parameter file for the new Replicat group.

```
EDIT PARAMS group
```

 **Note:**

You can copy the original parameter file to use for this group, but make certain to change the Replicat group name and any other relevant parameters that apply to this new group.

4. Add MAP statements (or edit copied ones) to specify the tables that you are adding or moving to this group. If this group will be a coordinated Replicat group, include the appropriate thread specifications.

5. Save and close the parameter file.
6. On the source system, run GGSCI.
7. Stop the Extract group.

```
STOP EXTRACT group
```

8. Issue the `INFO REPLICAT` command for the old Replicat group, and continue issuing it until it reports `At EOF, no more records to process`.

```
INFO REPLICAT group
```

9. On the target system, edit the old Replicat parameter file to remove `MAP` statements that specified the tables that you moved to the new Replicat group. Keep only the `MAP` statements that this Replicat will continue to process.

10. Save and close the file.

11. Issue the `INFO REPLICAT` command for the old Replicat group, and continue issuing it until it reports `At EOF, no more records to process`.

```
INFO REPLICAT group
```

12. Obtain the current Replicat checkpoint.

```
INFO REPLICAT group
```

13. Stop the old Replicat group. If you are stopping a coordinated Replicat, make certain the stop is clean so that all threads stop at the same trail record.

```
STOP REPLICAT group
```

14. Alter the new Replicat to position at the same trail sequence number and RBA as the old replicat group

```
ALTER REPLICAT group, EXTSEQNO seqno, EXTRBA rba
```

The *seqno* is the trail sequence number from the old group checkpoint obtained in step 11 and the *rba* is the trail record RBA number from the old group checkpoint.

15. Add the new Replicat group. For `EXTTRAIL`, specify the trail that this Replicat group is to read.

```
ADD REPLICAT group, EXTTRAIL trail
```

For example:

```
ADD REPLICAT rep, EXTTRAIL dirdat/rt
```

16. Issue the `INFORM COMMAND` to alter the Replicat to the trail file sequence number and RBA displayed.

```
INFORM COMMAND
```

17. On the source system, start the Extract group.

```
START EXTRACT group
```

18. On the target system, start the old Replicat group.

```
START REPLICAT group
```

19. Start the new Replicat group.

```
START REPLICAT group
```


Changing the Size of Trail Files

You can change the size of trail files with the `MEGABYTES` option of either the `ALTER EXTTRAIL` or `ALTER RMTTRAIL` command, depending on whether the trail is local or remote. To change the file size, follow this procedure.

1. Issue one of the following commands, depending on the location of the trail, to view the path name of the trail you want to alter and the name of the associated Extract group. Use a wildcard to view all trails.

(Remote trail)

```
INFO RMTTRAIL *
```

(Local trail)

```
INFO EXTTRAIL *
```

2. Issue one of the following commands, depending on the location of the trail, to change the file size.

(Remote trail)

```
ALTER RMTTRAIL trail, EXTRACT group, MEGABYTES n
```

(Local trail)

```
ALTER EXTTRAIL trail, EXTRACT group, MEGABYTES n
```

3. Issue the following command to cause Extract to switch to the next file in the trail.

```
SEND EXTRACT group, ROLLOVER
```

Switching Extract from Classic Mode to Integrated Mode

Valid for Oracle only.

This procedure switches an existing Extract group from classic mode to integrated mode. For more information about Extract modes for an Oracle database, see *Choosing Capture and Apply Modes in Using Oracle GoldenGate for Oracle Database*.

To support the transition to integrated mode, the transaction log that contains the start of the oldest open transaction must be available on the source or downstream mining system, depending on where Extract will be running.

To determine the oldest open transaction, issue the `SEND EXTRACT` command with the `SHOWTRANS` option. You can use the `FORCETRANS` or `SKIPTRANS` options of this command to manage specific open transactions, with the understanding that skipping a transaction may cause data loss and forcing a transaction to commit to the trail may add unwanted data if the transaction is rolled back by the user applications. Review these options in *SEND EXTRACT Reference for Oracle GoldenGate* before using them.

```
GGSCI> SEND EXTRACT group, SHOWTRANS
GGSCI> SEND EXTRACT group, { SKIPTRANS ID [THREAD n] [FORCE] |
FORCETRANS ID [THREAD n] [FORCE] }
```

To Switch Extract Modes

1. Back up the current Oracle GoldenGate working directories.

2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Extract parameter file to a new name.
3. Grant the appropriate privileges to the Extract user and perform the required configuration steps to support your business applications in integrated capture mode. See *Assigning Credentials to Oracle GoldenGate in Using Oracle GoldenGate for Oracle Database* for information about configuring and running Extract in integrated mode.
4. Log into the mining database with one of the following commands, depending on where the mining database is located.

```
DBLOGIN USERIDALIAS alias
```

```
MININGDBLOGIN USERIDALIAS alias
```

Where: *alias* specifies the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

5. Register the Extract group with the mining database. Among other things, this creates the logmining server.

```
REGISTER EXTRACT group DATABASE
```

6. Issue the following command to determine whether the upgrade command can be issued. Transactions that started before the registration command must be written to the trail before you can proceed with the upgrade. You may have to issue this command more than once until it returns a message stating that Extract can be upgraded.

```
INFO EXTRACT group UPGRADE
```

7. Stop the Extract group.

```
STOP EXTRACT group
```

8. Switch the Extract group to integrated mode. See Oracle RAC options for this command in `STOP EXTRACT` in *Reference for Oracle GoldenGate*, if applicable.

```
ALTER EXTRACT group UPGRADE INTEGRATED TRANLOG
```

9. Replace the old parameter file with the new one, keeping the same name.

10. Start the Extract group.

```
START EXTRACT group
```

Switching Extract from Integrated Mode to Classic Mode

Valid for Oracle only.

This procedure switches an existing Extract group from integrated mode to classic mode. For more information about Extract modes for an Oracle database, see *Choosing Capture and Apply Modes in Using Oracle GoldenGate for Oracle Database*.

To support the transition to classic mode, the transaction log that contains the start of the oldest open transaction must be available on the source or downstream mining system. To determine the oldest open transaction, issue the `SEND EXTRACT` command with the `SHOWTRANS` option. You can use the `FORCETRANS` or `SKIPTRANS` options of this

command to manage specific open transactions, with the understanding that skipping a transaction may cause data loss and forcing a transaction to commit to the trail may add unwanted data if the transaction is rolled back by the user applications. Review these options in Oracle GoldenGate Parameters in *Reference for Oracle GoldenGate* before using them.

```
GGSCI> SEND EXTRACT group, SHOWTRANS
GGSCI> SEND EXTRACT group, { SKIPTRANS ID [THREAD n] [FORCE] |
FORCETRANS ID [THREAD n] [FORCE] }
```

To Switch Extract Modes

1. Back up the current Oracle GoldenGate working directories.
2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Extract parameter file to a new name.
3. Grant the appropriate privileges to the Extract user and perform the required configuration steps to support your business applications in classic capture mode. See *Assigning Credentials to Oracle GoldenGate* in *Using Oracle GoldenGate for Oracle Database* for information about configuring and running Extract in classic mode.
4. Issue the following command to determine whether the downgrade command can be issued. Transactions that started before the downgrade command is issued must be written to the trail before you can proceed. You may have to issue this command more than once until it returns a message stating that Extract can be downgraded.

```
INFO EXTRACT group DOWNGRADE
```

5. Stop the Extract group.

```
STOP EXTRACT group
```

6. Log into the mining database with one of the following commands, depending on where the mining database is located.

```
DBLOGIN USERIDALIAS alias
```

```
MININGDBLOGIN USERIDALIAS alias
```

Where: *alias* is the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

7. Switch the Extract group to classic mode.

```
ALTER EXTRACT group DOWNGRADE INTEGRATED TRANLOG
```

If on a RAC system, then the `THREADS` option has to be used with the downgrade command to specify the number of RAC threads.

8. Unregister the Extract group from the mining database. Among other things, this removes the logmining server.

```
UNREGISTER EXTRACT group DATABASE
```

9. Replace the old parameter file with the new one, keeping the same name.
10. Start the Extract group.

```
START EXTRACT group
```

Switching Replicat from Nonintegrated Mode to Integrated Mode

Valid for Oracle only. For more information about Replicat modes for an Oracle database, see Choosing Capture and Apply Modes in *Using Oracle GoldenGate for Oracle Database*.

This procedure switches an existing Replicat group from nonintegrated to integrated mode.



Note:

Do not configure the switch between Replicat modes to occur immediately after Extract recovers from a failure or is repositioned to a different location in the transaction log.

1. Back up the Oracle GoldenGate working directories.
2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Replicat parameter file to a new name.
3. Grant the appropriate privileges to the Replicat user and perform the required configuration steps to support your business applications in integrated Replicat mode. See *Assigning Credentials to Oracle GoldenGate in Using Oracle GoldenGate for Oracle Database* for information about configuring and running Replicat in integrated mode.
4. Run GGSCI.
5. Stop Replicat.

```
STOP REPLICAT group
```

6. Log into the target database from GGSCI.

```
DBLOGIN USERIDALIAS alias
```

Where: *alias* is the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

7. Alter Replicat to integrated mode.

```
ALTER REPLICAT group, INTEGRATED
```

8. Replace the old parameter file with the new one, keeping the same name.
9. Start Replicat.

```
START REPLICAT group
```

10. Verify that Replicat is in integrated mode.

```
INFO REPLICAT group
```

When you start Replicat in integrated mode for the first time, the `START` command registers the Replicat group with the database and starts an inbound server to which Replicat attaches. When you convert a Replicat group to integrated mode, the use of the Oracle GoldenGate checkpoint table is discontinued and recovery information is maintained internally by the inbound server and by the checkpoint file going forward. You can retain the checkpoint table in the event that you decide to switch back to nonintegrated mode.

Switching Replicat from Integrated Mode to Nonintegrated Mode

Valid for Oracle only. For more information about Replicat modes for an Oracle database, see About Integrated Replicat in *Using Oracle GoldenGate for Oracle Database*.

You can, at any time, switch Replicat from integrated mode to nonintegrated mode. This switch automatically unregisters the Replicat group from the target database, which removes the inbound server.

Note:

Do not configure the switch between Replicat modes to occur immediately after Extract recovers from a failure or is repositioned to a different location in the transaction log.

Bug 17079228

1. Back up the Oracle GoldenGate working directories.
2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Replicat parameter file to a new name.
3. Grant the appropriate privileges to the Replicat user and perform the required configuration steps to support your business applications in nonintegrated Replicat mode. See *Assigning Credentials to Oracle GoldenGate* in *Using Oracle GoldenGate for Oracle Database* for information about configuring and running Replicat in integrated mode.
4. Run GGSCI.
5. Log into the target database from GGSCI.

```
DBLOGIN USERIDALIAS alias
```

Where: *alias* is the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

6. Create a checkpoint table in the target database for the nonintegrated Replicat to use to store its recovery checkpoints. If a checkpoint table was previously associated with this Replicat group and still exists, you can omit this step. See [Creating a Checkpoint Table](#) for more information about options for using a checkpoint table.

```
ADD CHECKPOINTTABLE [container.]table
```

7. Stop Replicat.

```
STOP REPLICAT group
```

8. Alter Replicat to nonintegrated mode. For the `CHECKPOINTTABLE` argument, specify the checkpoint table that you created for this Replicat group.

```
ALTER REPLICAT group, NONINTEGRATED, CHECKPOINTTABLE [container.]table
```

9. Replace the old parameter file with the new one, keeping the same name.

10. Start Replicat.

```
START REPLICAT group
```

After issuing this command, wait until there is some activity on the source database so that the switchover can be completed. (Replicat waits until its internal high-water mark is exceeded before removing the status of "switching from integrated mode.")

11. Verify that Replicat switched to nonintegrated mode.

```
INFO REPLICAT group
```

Switching Replicat to Coordinated Mode

Valid for all database types supported by Oracle GoldenGate.

This procedure upgrades a regular Replicat configuration (non-coordinated) to a coordinated configuration. This procedure assumes you are replacing a configuration that partitions data across multiple Extract and Replicat processes with a configuration that uses one Extract and one coordinated Replicat. The coordinated Replicat replaces the need for using multiple Replicat processes. A coordinated Replicat requires only one trail, so there is no need for multiple Extract processes or data pumps.

See [Configuring Online Change Synchronization](#) for more information about coordinated Replicat.

- [Procedure Overview](#)
- [Performing the Switch to Coordinated Replicat](#)

Procedure Overview

This procedure makes use of the `EVENTACTIONS` parameter with a `STOP` action, which enables all of the Replicat processes to stop at the same point in the trail. The `EVENTACTIONS` action is triggered by a transaction that contains an `INSERT` to a dummy table. The `INSERT` causes each process to finish processing everything up to, and including, the event transaction and then stop cleanly. An additional event action of `IGNORE` is specified for Replicat to prevent the multiple Replicat processes from attempting to insert the same record to the target. The result of this procedure is that all processes stop at the same point in the data stream: after completing the `INSERT` transaction to the dummy table.

After the processes stop, you move all of the `TABLE` statements to one primary Extract group. You move the same `TABLE` statements to the data pump that reads the trail of the Extract group that you retained. You move all of the `MAP` statements to a new

coordinated Replicat group that reads the remote trail that is associated with the retained data pump. Once all of the `MAP` statements are in one parameter file, you edit them to add the thread specifications to support a coordinated Replicat. (This can be done ahead of time.) Then you drop the Replicat group and add it back in coordinated mode with the same name.

Performing the Switch to Coordinated Replicat

Note:

Do not create the Replicat group until prompted by these instructions.

1. Back up the current parameter files of all of the Extract groups, data pumps, and Replicat groups. You will be editing them.
2. Create a working directory outside the Oracle GoldenGate directory. You will use this directory to create and stage new versions of the parameter files. If needed, you can create a working directory on the source and target systems.
3. In the working directory, create a parameter file for a coordinated Replicat. Copy the `MAP` parameters from the active parameter files of all of the Replicat groups to this parameter file, and then add the thread specifications and any other parameters that support your required coordinated Replicat configuration.
4. If using multiple primary Extract groups, select one to keep, and then save a copy of its current parameter file to the working directory.
5. Copy all of the `TABLE` statements from the other Extract groups to the new parameter file of the primary Extract that you are keeping.
6. In the working directory, save a copy of the parameter file of the data pump that is linked to the primary Extract that you are keeping.
7. Copy all of the `TABLE` statements from the other data pumps to the copied parameter file of the kept data pump.
8. In the source database, create a simple dummy table on which a simple `INSERT` statement can be performed. For this procedure, the name `schema.event` is used.
9. Create the same table on the target system, to avoid the need for additional configuration parameters.
10. Edit the active parameter files (not the copies) of all primary and data-pump Extract groups to add the following `EVENTACTIONS` parameter to each one.

```
TABLE schema.event, EVENTACTIONS(STOP);
```

11. Edit the active parameter files (not the copies) of all of the Replicat groups to add the following `EVENTACTIONS` parameter to each one.

```
MAP schema.event, TARGET schema.event, EVENTACTIONS(IGNORE, STOP);
```

12. Stop the Oracle GoldenGate processes gracefully in the following order:
 - Stop all Replicat processes.
 - Stop all data pumps.
 - Stop all Extract processes.

13. Restart the Oracle GoldenGate processes in the following order so that the `EVENTACTIONS` parameters take effect:
 - Start all Extract processes.
 - Start all data pumps.
 - Start all Replicat processes.
14. On the source system, issue a transaction on the `schema.event` table that contains one `INSERT` statement. Make certain to commit the transaction.
15. In GGSCI, issue the `STATUS` command for all of the primary Extract and data pump processes on the source system, and issue the same command for all of the Replicat processes on the target system, until the commands show that all of the processes are `STOPPED`.

```
STATUS EXTRACT *  
STATUS REPLICAT *
```

16. Replace the active parameter files of the primary Extract and data pump that you kept with the new parameter files from the working directory.
17. Delete the unneeded Extract and data pump groups and their parameter files.
18. Log into the target database by using the `DBLOGIN` command.
19. Delete all of the Replicat groups and their active parameter files.
20. Copy or move the new coordinated Replicat parameter file from the working directory to the Oracle GoldenGate directory.
21. In GGSCI, issue the `INFO EXTRACT` command for the data pump and make note of its write checkpoint position in the output (remote) trail.

```
INFO EXTRACT pump, DETAIL
```

22. Add a new coordinated Replicat group with the following parameters.

```
ADD REPLICAT group, EXTTRAIL trail, EXTSEQNO sequence_number, EXTRBA rba,  
COORDINATED MAXTHREADS number
```

Where:

- `group` is the name of the coordinated Replicat group. The name must match that of the new parameter file created for this group.
 - `EXTTRAIL trail` is the name of the trail that the data pump writes to.
 - `EXTSEQNO sequence_number` is the sequence number of the trail as shown in the write checkpoint returned by the `INFO EXTRACT` that you issued for the data pump.
 - `EXTRBA rba` is the relative byte address in the trail as shown in the write checkpoint returned by `INFO EXTRACT`. Together, these position Replicat to resume processing at the correct point in the trail.
 - `MAXTHREADS number` specifies the maximum number of threads allowed for this group. This value should be appropriate for the number of threads that are specified in the parameter file.
23. Start the primary Extract group.
 24. Start the data pump group.
 25. Start the coordinated Replicat group.

Administering a Coordinated Replicat Configuration

This section contains instructions for coordinating threads and re-partitioning the workload among new or different threads. A coordinated Replicat should be stopped cleanly with the `STOP REPLICAT` command before making modifications to the partition specifications in `THREAD` or `THREADRANGE` clauses of the `MAP` statements. A clean stop ensures that all of the threads, which may be at different locations in the trail at any given point, all finish their work and arrive at a common trail location.

At startup, Replicat issues an error and abends if it detects that the last shutdown was not clean and the partitioning in the `MAP` statements was changed to contain a different number of threads (threads were added or removed). However, if the same threads are kept in the parameter file but simply rearranged among different `MAP` statements, Replicat issues a warning but does not abend. This can result in missing or duplicate records, because there is no way to ensure continuity of the thread-to-workload allocations from the previous run.

The following is an example of this condition.

Following is the original partitioning scheme:

```
MAP source, target, THREADRANGE(1-5);  
MAP source1, target1, THREADRANGE(6-10);
```

The following re-partitioning of the original scheme produces only a warning:

```
MAP source, target, THREADRANGE(1-4);  
MAP source1, target1, THREADRANGE(5-10);
```

This section provides instructions for cleanly shutting down Replicat before performing a re-partitioning, as well as instructions for attempting to recover Replicat continuity when a re-partitioning is performed after an unclean shutdown.

The following tasks can be performed for a Replicat group in coordinated mode.

- [Performing a Planned Re-partitioning of the Workload](#)
- [Recovering Replicat After an Unplanned Re-partitioning](#)
- [Synchronizing Threads After an Unclean Stop](#)

Performing a Planned Re-partitioning of the Workload

A planned re-partitioning is when Replicat is allowed to shut down cleanly before it is started again with a new parameter file that contains updated thread partitioning. A clean shutdown enables all of the threads to arrive at a common checkpoint position in the trail. At that point, the new partitioning scheme can be applied in the next run. If Replicat does not shut down cleanly in this procedure, for example if there is an apply error, use the procedure in [Synchronizing Threads After an Unclean Stop](#) to re-synchronize the threads before you re-partition them.

1. Run GGSCI.
2. Stop Replicat.

```
STOP REPLICAT group
```
3. Open the parameter file for editing.

```
EDIT PARAMS group
```

4. Make the required changes to the `THREAD` or `THREADRANGE` specifications in the `MAP` statements.
5. Save and close the parameter file.
6. Start Replicat.

```
START REPLICAT group
```

Recovering Replicat After an Unplanned Re-partitioning

An *unplanned re-partitioning* is when Replicat is not allowed to shut down cleanly before it is started again with a new parameter file that contains updated thread partitioning. In this scenario, some or all of the old threads were not able to finish their work and arrive at a common checkpoint. Upon restart, the coordinator thread attempts to apply the old partitioning scheme, and Replicat abends with an error. You can recover the coordinated Replicat group from this condition in one of the following ways:

- Use the auto-saved copy of the parameter file
- Reprocess from the low watermark with `HANDLECOLLISIONS`
- [Reprocessing From the Low Watermark with HANDLECOLLISIONS](#)
- [Using the Auto-Saved Parameter File](#)

Reprocessing From the Low Watermark with HANDLECOLLISIONS

In this procedure, you reposition all of the threads to the *low watermark* position. This is the earliest checkpoint position performed among all of the threads. To state it another way, the low watermark position is the last record processed by the slowest thread before the unclean stop. When you start Replicat, the threads reprocess the operations that they were processing before Replicat stopped, and the `HANDLECOLLISIONS` parameter handles any duplicate-record and missing-record errors that occur as the faster threads reprocess operations that they applied before the unclean stop.

1. Add the `HANDLECOLLISIONS` parameter to the Replicat parameter file. It is not necessary to use any `THREADS` options.
2. Issue the `INFO REPLICAT` command for the Replicat group as a whole (the coordinator thread). Make a record of the RBA of the checkpoint. This is the *low watermark* value. This output also shows you the active thread IDs under the `Group Name` column. Make a record of these, as well.

```
INFO REPLICAT group
```

```
GGSCI (slc03jgo) 3> info ra detailREPLICAT RA      Last Started 2013-05-01
14:15  Status ABENDEDCOORDINATED      Coordinator
MAXTHREADS 15Checkpoint Lag      00:00:00 (updated 00:00:07 ago)Process
ID      11445Log Read Checkpoint File ./dirdat/withMaxTransOp/
bg000000001      2013-05-02 07:49:45.975662 RBA 44704Lowest Log
BSN value: (requires database login)Active Threads: ID Group Name PID
Status Lag at Chkpt Time Since Chkpt1 RA001 11454 ABENDED
00:00:00 00:00:01 2 RA002 11455 ABENDED 00:00:00 00:00:04
3 RA003 11456 ABENDED 00:00:00 00:00:01 5 RA005 11457
```

```

ABENDED 00:00:00      00:00:02      6  RA006      11458 ABENDED 00:00:00
00:00:04      7  RA007      11459 ABENDED 00:00:00      00:00:04

```

3. Issue the `INFO REPLICAT` command for each processing thread ID and record the RBA position of each thread. Make a note of the *highest* RBA. This is the *high watermark* of the Replicat group.

```
INFO REPLICAT threadID
```

```

info ra002
REPLICAT  RA002      Last Started 2013-05-01 14:15      Status
ABENDEDCOORDINATED      Replicat Thread      Thread 2Checkpoint
Lag      00:00:00 (updated 00:00:06 ago)Process ID      11455
Log Read Checkpoint File ./dirdat/withMaxTransOp/
bg000000001      2013-05-02 07:49:15.837271 RBA 45603

```

4. Issue the `ALTER REPLICAT` command for the coordinator thread (Replicat as a whole, without any thread ID) and position to the *low watermark* RBA that you recorded.

```
ALTER REPLICAT group EXTRBA low_watermark_rba
```

5. Start Replicat.

```
START REPLICAT group
```

6. Issue the basic `INFO REPLICAT` command until it shows an RBA that is higher than the *high watermark* that you recorded. `HANDLECOLLISIONS` handles any collisions that occur due to previously applied transactions.

```
INFO REPLICAT group
```

7. Stop Replicat.

```
STOP REPLICAT group
```

8. Remove or comment out the `HANDLECOLLISIONS` parameter.

9. Start Replicat.

```
START REPLICAT group
```

Using the Auto-Saved Parameter File

A copy of the original parameter file is saved whenever the parameter file is edited before shutting down Replicat cleanly. You can revert to this parameter file and then resynchronize the threads so that they all catch up to the thread that had the most recent checkpoint position. Once the threads are synchronized, you can switch to the new parameter file and then start Replicat.

1. Save the new parameter file to a different name, and then rename the saved original parameter file to the correct name (same as the group name). The saved parameter file has a `.backup` suffix and is stored in the `dirprm` subdirectory of the Oracle GoldenGate installation directory.
2. Issue the following command to synchronize the Replicat threads to the maximum checkpoint position. This command automatically starts Replicat and executes the threads until they reach the maximum checkpoint position.

```
SYNCHRONIZE REPLICAT group
```

3. Issue the `STATUS REPLICAT` command until it shows that Replicat stopped cleanly.

```
STATUS REPLICAT group
```

4. Save the original parameter file to a different name, and then rename the new parameter file to the group name.
5. Start Replicat.

```
START REPLICAT group
```

Synchronizing Threads After an Unclean Stop

When a Replicat group stops in an unclean manner, not all of the threads will reach a common checkpoint position in the trail. Unclean stops can be caused by issuing `STOP REPLICAT` with the `!` option, issuing the `KILL REPLICAT` command, or by transient errors related to Replicat, the database, or other local processes. You can restore the threads to the same position in the trail after an unclean stop and then start Replicat again from the correct checkpoint position.

In this procedure, the restore position is the *high watermark*. This is the most recent checkpoint position performed among all of the threads (the last record processed by the fastest thread before the unclean stop). Before starting Replicat, you can make changes to the parameter file, such as to repartition the workload among different or new threads. The repartitioning takes effect in a seamless manner after you start Replicat, because the threads can start from a synchronized state.

1. Run GGSCI.
2. Synchronize the Replicat threads to the maximum checkpoint position. Replicat performs the synchronization and then stops.

```
SYNCHRONIZE REPLICAT group
```

3. (Optional) To re-partition the workload among different or new threads, open the parameter file for editing and then make the required changes to the `THREAD` or `THREADRANGE` specifications in the `MAP` statements.

```
EDIT PARAMS group
```

4. Save and close the parameter file.
5. Start Replicat.

```
START REPLICAT group
```

Restarting a Primary Extract after System Failure or Corruption

This procedure enables Oracle GoldenGate to recover from certain conditions, such as a file system corruption or a system failure, that corrupt the Extract checkpoint file, trail, or both, and which prevent Extract from being able to start. It enables you to establish a safe starting point in the transaction log for the primary Extract after the system has been restored. It also shows you how to reposition downstream data pumps and Replicat to read from the correct Extract write position in the trails, and to filter out any transactions that Replicat already applied to the target.

- [Details of This Procedure](#)
- [Performing the Recovery](#)

Details of This Procedure

Extract passes a *log begin sequence number*, or *LOGBSN*, to the trail files. The BSN is the native database sequence number that identifies the oldest uncommitted transaction that is held in Extract memory. For example, the BSN in an Oracle installation would be the Oracle system change number (SCN). Each trail file contains the lowest *LOGBSN* value for all of the transactions in that trail file. Once you know the *LOGBSN* value, you can reposition Extract at the correct read position to ensure that the appropriate transactions are re-generated to the trail and propagated to Replicat.

 **Note:**

In an Oracle RAC environment, the lowest SCN of all of the threads is transmitted to Replicat. Transactions that may already have been committed by Replicat are handled as duplicates at startup. However, any thread that has been idle past a certain threshold will not be considered for the BSN value, to avoid Extract having to read too far back in the log stream when restarted.

The bounded recovery checkpoint is not taken into account when calculating the *LOGBSN*. The failure that affected the Extract checkpoint file may also involve a loss of the persisted bounded recovery data files and bounded recovery checkpoint information.

Performing the Recovery

Follow these steps in the order shown to recover the Oracle GoldenGate processes.

1. In GGSCI on the target system, issue the `DBLOGIN` command.

```
DBLOGIN {USERID Replicat_user | USERIDALIAS alias_of_Replicat_user}
```

2. On the target, obtain the *LOGBSN* value by issuing the `INFO REPLICAT` command with the `DETAIL` option.

```
INFO REPLICAT group, DETAIL
```

The BSN is included in the output as a line similar to the following:

```
Current Log BSN value: 1151679
```

3. (Classic capture mode only. Skip if using integrated capture mode.) Query the source database to find the sequence number of the transaction log file that contains the value of the *LOGBSN* that you identified in the previous step. This example assumes 1855798 is the *LOGBSN* value and shows that the sequence number of the transaction log that contains that *LOGBSN* value is 163.

```
SQL> select name, thread#, sequence# from v$archived_log  
where 1855798 between first_change# and next_change#;
```

NAME	THREAD#	SEQUENCE#
-----/oracle/dbs/ arch1_163_800262442.dbf	1	163

4. Issue the following commands in GGSCI to reposition the primary Extract to the LOGBSN start position.

- (Classic capture mode)

```
ALTER EXTRACT group EXTSEQNO 163
ALTER EXTRACT group EXTRBA 0
ALTER EXTRACT group ETROLLOVER
```

- (Integrated capture mode)

```
ALTER EXTRACT group SCN 1151679
ALTER EXTRACT group ETROLLOVER
```

 **Note:**

There is a limit on how far back Extract can go in the transaction stream, when in integrated mode. If the required SCN is no longer available, the ALTER EXTRACT command fails.

5. Issue the following command in GGSCI to the primary Extract to view the new sequence number of the Extract *Write Checkpoint*. This command shows the trail and RBA where Extract will begin to write new data. Because a rollover was issued, the start point is at the beginning (RBA 0) of the new trail file, in this example file number 7.

```
INFO EXTRACT group SHOWCH
Sequence #: 7
RBA: 0
```

6. Issue the following command in GGSCI to reposition the downstream data pump and start a new output trail file.

```
ALTER EXTRACT pump EXTSEQNO 7
ALTER EXTRACT pump EXTRBA 0
ALTER EXTRACT pump ETROLLOVER
```

7. Issue the following command in GGSCI to the data pump Extract to view the new sequence number of the data pump Write Checkpoint, in this example trail number 9.

```
INFO EXTRACT pump SHOWCH
Sequence #: 9
RBA: 0
```

8. Reposition Replicat to start reading the trail at the new Write Checkpoint of the data pump.

```
ALTER REPLICAT group EXTSEQNO 9
ALTER REPLICAT group EXTRBA 0
```

9. Start the primary Extract and the data pump.

```
START EXTRACT group
START REPLICAT group
```

10. Issue the following command in GGSCI to start Replicat. If Replicat is operating in integrated mode (Oracle targets only), you do not need the FILTERDUPTRANSACTIONS option. Integrated Replicat handles duplicate transactions transparently.

```
START REPLICAT group[, FILTERDUPTRANSACTIONS]
```

 **Note:**

The LOGBSN gives you the information needed to set Extract back in time to reprocess transactions. Some filtering by Replicat is necessary because Extract will likely re-generate a small amount of data that was already applied by Replicat. FILTERDUPTRANSACTIONS directs Replicat to find and filter duplicates at the beginning of the run.

Part II

Administering Oracle GoldenGate Microservices Architecture

The Oracle GoldenGate MA provides all the tools you need to configure, monitor, and administer deployments and security. It is designed with the industry-standard HTTP(s) communication protocol and the JavaScript Object Notation (JSON) data interchange format. In addition, the architecture provides you with the ability to verify the identity of clients with basic authentication or Secure Sockets Layer client certificates.

- [Loading Data from File to Replicat in Microservices Architecture](#)
By following the steps provided in this topic, data can be precisely replicated from a source to a target database with zero data loss using a combination of file-based initial load and change data capture (CDC) processes.

Loading Data from File to Replicat in Microservices Architecture

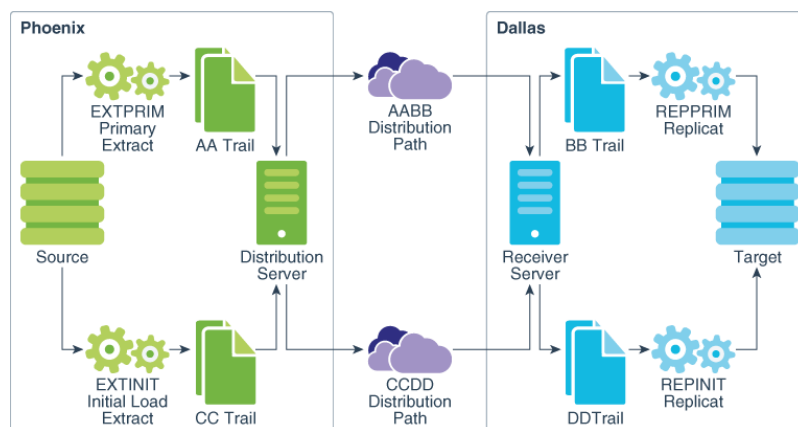
By following the steps provided in this topic, data can be precisely replicated from a source to a target database with zero data loss using a combination of file-based initial load and change data capture (CDC) processes.

In [Loading Data from File to Replicat](#), the initial load process is implemented using files. However, Microservices Architecture uses a different approach. The process of creating and running a replication solution constitutes:

- Initial Load: Used to copy the existing contents of one or more tables from the source to the target database.
- Change Data Capture: Used to copy transactional changes from the source to the target database.

Note:

MA doesn't support loading data with an Oracle GoldenGate direct load.



File-based initial load process is the preferred method for performing data replication in MA. Its key components are:

- Initial Load Extract and Replicat: Replicates the existing content of the database tables.
- Primary Extract and Replicat: Replicates change data from the database tables.
- Distribution Paths: Transfers trail files to the target system.

Before you begin, make sure that the database credential alias is created.

! Important:

This topic demonstrates the steps for initial load processing using the AdminClient. However, you can also use curl to perform these steps.

Step 1: Creating a Primary Extract

Precise instantiation is used to replicate database resources correctly from the source to the target database. The primary Extract is started first to initiate change data capture early. Precise instantiation is based on the following assumptions:

Note:

For precise instantiation to work, the instantiation SCN must come after the registration SCN.

- The primary Extract is started. It is responsible for change data capture and noting its registration SCN.
- The database is monitored. The database waits for the oldest open transaction's SCN to come after the registration SCN. This is the instantiation SCN.
- The instantiation SCN is used when creating the initial load Extract and Replicat processes.
- The instantiation SCN is used to create the primary Replicat, once the initial load replication is complete.

To begin, create and start the primary Extract `EXTPRIM` from the AdminClient, as shown in the following example:

```
OGG (not connected) 1> connect https://phoenix.oggdevops.us:9100 as
oggadmin password oggadmin !
Using default deployment 'Phoenix'

OGG (https://phoenix.oggdevops.us:9100 Phoenix) 2> dblogin useridentialias
oggadmin
Successfully logged into database.

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 3> add extract
EXTPRIM integrated tranlog begin now
2018-03-16T13:37:07Z INFO OGG-08100 EXTRACT (Integrated) added.

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 4> register
extract EXTPRIM database
2018-03-16T13:37:30Z INFO OGG-02003 Extract EXTPRIM successfully
registered with database at SCN 1608891.

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 5> edit params
EXTPRIM

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 6> view params
EXTPRIM
```

```

--
--  E X T P R I M . p r m
--  Primary Extract Parameter File
--
Extract      EXTPRIM
UseridAlias  oggadmin
ExtTrail     AA
Table        user01.*;

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 7> add
exttrail AA extract EXTPRIM
2018-03-16T13:37:55Z  INFO      OGG-08100  EXTTRAIL added.

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 8> start
extract EXTPRIM
2018-03-16T13:38:02Z  INFO      OGG-00975  EXTRACT EXTPRIM starting
2018-03-16T13:38:02Z  INFO      OGG-15426  EXTRACT EXTPRIM started

```

In this example, oggadmin is the database credential alias.

After creating the primary Extract, retrieve the SCN registration number. Run the REGISTER EXTRACT command in the AdminClient. The following example retrieves an SCN value of 1608891.

```

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 4> register
extract EXTPRIM database
2018-03-16T13:37:30Z  INFO      OGG-02003  Extract EXTPRIM successfully
registered with database at SCN 1608891.

```

Step 2: Determining the Instantiation SCN

The Administration Server exposes an endpoint that returns details of all active transactions and the current database SCN. The PL/SQL query to retrieve this data is:

```

-- Query for active transactions
--
Select T.START_SCN, T.STATUS TSTATUS, T.START_DATE,
       S.SID, S.SERIAL#, S.INST_ID, S.USERNAME, S.OSUSER, S.STATUS
SSTATUS, S.LOGON_TIME
  From gv$transaction T
  Inner
  Join gv$session S
  on S.SADDR = T.SES_ADDR

Union All

--
-- Query for current status
--
Select current_scn, 'CURRENT', CURRENT_DATE,
       NULL, NULL, NULL, 'SYS', NULL, NULL, NULL
  from v$database

Order by 1;

```

The results of this query can be used to determine the instantiation SCN. The results for this specific query are:

```
1538916          ACTIVE          2018-03-16 18:10:31.0          3865
9176            1          GGADMIN          oracle          INACTIVE          2018-03-16
18:10:26.0 1540555          CURRENT          2018-03-16
18:21:50.0                                     SYS
```

The SCN used to instantiate the initial load Extract is obtained using SQL*Plus. In the following example, the SQL query uses the instantiation SCN value as 1624963, which is the oldest SCN of all open transactions that are also past the registration SCN of 1608891.

```
OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 14> shell echo
'Select MIN(START_SCN) From gv$transaction;' | ${ORACLE_HOME}/bin/sqlplus -
S / as sysdba
```

```
MIN(START_SCN)
-----
              1624963
```

If there are no open transactions, then this SQL query returns an empty result. A detailed query that takes into account the situation where there are no open transactions is:

```
Select MIN(SCN) as INSTANTIATION_SCN
  From (Select MIN(START_SCN) as SCN
        From gv$transaction
        Union All
        Select current_scn
        From gv$database);
```

Step 3: Creating and Starting the Initial Load Replicat

Before you begin this step, make sure that the checkpoint table `oggadmin.checkpoints`, already exists on the target system. The initial load Replicat is responsible for populating the target database. Run the following command on the AdminClient to create and start the initial load Replicat (`REPINIT`):

```
OGG (not connected) 1> connect https://dallas.oggdevops.us:9100 as
oggadmin password oggadmin !
Using default deployment 'Dallas'
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas) 2> dblogin useridalias
oggadmin
Successfully logged into database.
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 3> add
checkpointtable oggadmin.checkpoints
ADD "oggadmin.checkpoints" succeeded.
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 4> add replicat
REPINIT exttrail DD checkpointtable oggadmin.checkpoints
```

```

2018-03-16T13:56:41Z INFO OGG-08100 REPLICAT added.

OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 5> edit params
REPINIT

OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 6> view params
REPINIT
--
-- R E P I N I T . p r m
-- File-Based Initial Load Replicat Parameter File
--
Replicat      REPINIT
UseridAlias   oggadmin
Map           user01.*
Target        user01.*;

OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 7> start
replicat REPINIT
2018-03-16T13:58:21Z INFO OGG-00975 REPLICAT REPINIT starting
2018-03-16T13:58:21Z INFO OGG-15426 REPLICAT REPINIT started

```

Step 4: Creating and starting the Initial Load Extract

Using the instantiation SCN that you retrieved (1624963), the initial load Extract is created to write contents of the database tables to the trail.

Create and start the initial load extract, EXTINIT.

```

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 15> add
extract EXTINIT sourceistable
2018-03-16T14:08:38Z INFO OGG-08100 EXTRACT added.

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 16> edit
params EXTINIT

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 17> view
params EXTINIT
--
-- E X T I N I T . p r m
-- File-Based Initial Load Extract Parameter File
--
Extract      EXTINIT
UseridAlias   oggadmin
ExtFile       CC Megabytes 2000 Purge
Table         user01.*, SQLPredicate "As Of SCN 1609723";

OGG (https://phoenix.oggdevops.us:9100 Phoenix as oggadmin) 18> start
extract EXTINIT
2018-03-16T14:13:42Z INFO OGG-00975 EXTRACT EXTINIT starting
2018-03-16T14:13:42Z INFO OGG-15426 EXTRACT EXTINIT started

```

Step 5: Creating the Distribution Paths

Create two distribution paths (AABB and CCDD) for copying the local trails to the remote host from the AdminClient:

```
OGG (https://phoenix.oggdevops.us:9100 Phoenix) 15> add distpath AABB
source trail://phoenix.oggdevops.us:9102/services/v2/sources?trail=AA
target wss://dallas.oggdevops.us:9103/services/v2/targets?trail=BB
2018-03-16T17:28:27Z INFO OGG-08511 The path 'AABB' has been added.
```

```
OGG (https://phoenix.oggdevops.us:9100 Phoenix) 16> add distpath CCDD
source trail://phoenix.oggdevops.us:9102/services/v2/sources?trail=CC
target wss://dallas.oggdevops.us:9103/services/v2/targets?trail=DD
2018-03-16T17:28:35Z INFO OGG-08511 The path 'CCDD' has been added.
```

```
OGG (https://phoenix.oggdevops.us:9100 Phoenix) 17> start distpath AABB
2018-03-16T17:28:42Z INFO OGG-08513 The path 'AABB' has been started.
```

```
OGG (https://phoenix.oggdevops.us:9100 Phoenix) 18> start distpath CCDD
2018-03-16T17:28:47Z INFO OGG-08513 The path 'CCDD' has been started.
```

Step 6: Creating the Primary Replicat REPPRIM

Once the initial load Extract and Replicat complete, they can be deleted. Then, the primary Replicat process is created on the remote host for applying change data to the target database.

Use the AdminClient to create the primary Replicat process.



Note:

The primary Replicat is started at the instantiation SCN.

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 12> add replicat
REPPRIM exttrail BB checkpointtable oggadmin.checkpoints
2018-03-16T17:37:46Z INFO OGG-08100 REPLICAT added.
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 13> edit params
REPPRIM
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 14> view params
REPPRIM
```

```
--
-- R E P P R I M . p r m
-- Replicat Parameter File
--
Replicat REPPRIM
UseridAlias oggadmin
Map user01.*
Target user01.*;
```

```
OGG (https://dallas.oggdevops.us:9100 Dallas as oggadmin) 15> start
```

```
replicat REPRIM atcsn 1624963  
2018-03-16T17:38:10Z INFO OGG-00975 REPLICAT REPRIM starting  
2018-03-16T17:38:10Z INFO OGG-15426 REPLICAT REPRIM started
```

A

Supported Character Sets

This appendix lists the character sets that Oracle GoldenGate supports when converting data from source to target.

The identifiers that are shown should be used for Oracle GoldenGate parameters or commands when a character set must be specified, instead of the actual character set name. Currently Oracle GoldenGate does not provide a facility to specify the database-specific character set.

Topics:

- [Supported Character Sets - Oracle](#)
- [Supported Character Sets - Non-Oracle](#)

Supported Character Sets - Oracle

Table A-1 Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
ar8ados710t	Arabic MS-DOS 710 8-bit Latin/Arabic
ar8ados710	Arabic MS-DOS 710 Server 8-bit Latin/Arabic
ar8ados720t	Arabic MS-DOS 720 8-bit Latin/Arabic
ar8ados720	Arabic MS-DOS 720 Server 8-bit Latin/Arabic
ar8aptec715t	APTEC 715 8-bit Latin/Arabic
ar8aptec715	APTEC 715 Server 8-bit Latin/Arabic
ar8arabicmacs	Mac Server 8-bit Latin/Arabic
ar8arabicmact	Mac 8-bit Latin/Arabic
ar8arabicmac	Mac Client 8-bit Latin/Arabic
ar8asmo708plus	ASMO 708 Plus 8-bit Latin/Arabic
ar8asmo8x	ASMO Extended 708 8-bit Latin/Arabic
ar8ebcdic420s	EBCDIC Code Page 420 Server 8-bit Latin/Arabic
ar8ebcdicx	EBCDIC XBASIC Server 8-bit Latin/Arabic
ar8hparabic8t	HP 8-bit Latin/Arabic
ar8iso8859p6	ISO 8859-6 Latin/Arabic
ar8mswin1256	MS Windows Code Page 1256 8-Bit Latin/Arabic
ar8mussad768t	Mussa'd Alarabi/2 768 8-bit Latin/Arabic
ar8mussad768	Mussa'd Alarabi/2 768 Server 8-bit Latin/Arabic

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
ar8nafitha711t	Nafitha International 711 Server 8-bit Latin/Arabic
ar8nafitha711	Nafitha Enhanced 711 Server 8-bit Latin/Arabic
ar8nafitha721t	Nafitha International 721 8-bit Latin/Arabic
ar8nafitha721	Nafitha International 721 Server 8-bit Latin/Arabic
ar8sakhr706	SAKHR 706 Server 8-bit Latin/Arabic
ar8sakhr707t	SAKHR 707 8-bit Latin/Arabic
ar8sakhr707	SAKHR 707 Server 8-bit Latin/Arabic
ar8xbasic	XBASIC 8-bit Latin/Arabic
az8iso8859p9e	ISO 8859-9 Azerbaijani
bg8mswin	MS Windows 8-bit Bulgarian Cyrillic
bg8pc437s	IBM-PC Code Page 437 8-bit (Bulgarian Modification)
blt8cp921	Latvian Standard LVS8-92(1) Windows/Unix 8-bit Baltic
blt8ebcdic1112s	EBCDIC Code Page 1112 8-bit Server Baltic Multilingual
blt8ebcdic1112	EBCDIC Code Page 1112 8-bit Baltic Multilingual
blt8iso8859p13	ISO 8859-13 Baltic
blt8mswin1257	MS Windows Code Page 1257 8-bit Baltic
blt8pc775	IBM-PC Code Page 775 8-bit Baltic
bn8bscii	Bangladesh National Code 8-bit BSCII
cdn8pc863	IBM-PC Code Page 863 8-bit Canadian French
ce8bs2000	Siemens EBCDIC.DF.04-2 8-bit Central European
cel8iso8859p14	ISO 8859-13 Celtic
ch7dec	DEC VT100 7-bit Swiss (German/French)
cl8bs2000	Siemens EBCDIC.EHC.LC 8-bit Latin/Cyrillic-1
cl8ebcdic1025c	EBCDIC Code Page 1025 Client 8-bit Cyrillic
cl8ebcdic1025r	EBCDIC Code Page 1025 Server 8-bit Cyrillic
cl8ebcdic1025s	EBCDIC Code Page 1025 Server 8-bit Cyrillic
cl8ebcdic1025	EBCDIC Code Page 1025 8-bit Cyrillic
cl8ebcdic1025x	EBCDIC Code Page 1025 (Modified) 8-bit Cyrillic
cl8ebcdic1158r	EBCDIC Code Page 1158 Server 8-bit Cyrillic
cl8ebcdic1158	EBCDIC Code Page 1158 8-bit Cyrillic
cl8iso8859p5	ISO 8859-5 Latin/Cyrillic
cl8isoir111	SOIR111 Cyrillic
cl8koi8r	RELCOM Internet Standard 8-bit Latin/Cyrillic

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
cl8koi8u	KOI8 Ukrainian Cyrillic
cl8maccyrillics	Mac Server 8-bit Latin/Cyrillic
cl8maccyrillic	Mac Client 8-bit Latin/Cyrillic
cl8mswin1251	MS Windows Code Page 1251 8-bit Latin/Cyrillic
d7dec	DEC VT100 7-bit German
d7siemens9780x	Siemens 97801/97808 7-bit German
d8bs2000	Siemens 9750-62 EBCDIC 8-bit German
d8ebcdic1141	EBCDIC Code Page 1141 8-bit Austrian German
d8ebcdic273	EBCDIC Code Page 273/1 8-bit Austrian German
dk7siemens9780x	Siemens 97801/97808 7-bit Danish
dk8bs2000	Siemens 9750-62 EBCDIC 8-bit Danish
dk8ebcdic1142	EBCDIC Code Page 1142 8-bit Danish
dk8ebcdic277	EBCDIC Code Page 277/1 8-bit Danish
e7dec	DEC VT100 7-bit Spanish
e7siemens9780x	Siemens 97801/97808 7-bit Spanish
e8bs2000	Siemens 9750-62 EBCDIC 8-bit Spanish
ee8bs2000	Siemens EBCDIC.EHC.L2 8-bit East European
ee8ebcdic870c	EBCDIC Code Page 870 Client 8-bit East European
ee8ebcdic870s	EBCDIC Code Page 870 Server 8-bit East European
ee8ebcdic870	EBCDIC Code Page 870 8-bit East European
ee8iso8859p2	ISO 8859-2 East European
ee8maccess	Mac Server 8-bit Central European
ee8macce	Mac Client 8-bit Central European
ee8maccroatians	Mac Server 8-bit Croatian
ee8maccroatian	Mac Client 8-bit Croatian
ee8mswin1250	MS Windows Code Page 1250 8-bit East European
ee8pc852	IBM-PC Code Page 852 8-bit East European
eec8euroasci	EEC Targon 35 ASCII West European/Greek
eec8europa3	EEC EUROPA3 8-bit West European/Greek
el8dec	DEC 8-bit Latin/Greek
el8ebcdic423r	IBM EBCDIC Code Page 423 for RDBMS server-side
el8ebcdic875r	EBCDIC Code Page 875 Server 8-bit Greek
el8ebcdic875s	EBCDIC Code Page 875 Server 8-bit Greek

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
el8ebcdic875	EBCDIC Code Page 875 8-bit Greek
el8gc0s7	Bull EBCDIC GCOS7 8-bit Greek
el8iso8859p7	ISO 8859-7 Latin/Greek
el8macgreek	Mac Server 8-bit Greek
el8macgreek	Mac Client 8-bit Greek
el8mswin1253	MS Windows Code Page 1253 8-bit Latin/Greek
el8pc437s	IBM-PC Code Page 437 8-bit (Greek modification)
el8pc737	IBM-PC Code Page 737 8-bit Greek/Latin
el8pc851	IBM-PC Code Page 851 8-bit Greek/Latin
el8pc869	IBM-PC Code Page 869 8-bit Greek/Latin
et8mswin923	MS Windows Code Page 923 8-bit Estonian
f7dec	DEC VT100 7-bit French
f7siemens9780x	Siemens 97801/97808 7-bit French
f8bs2000	Siemens 9750-62 EBCDIC 8-bit French
f8ebcdic1147	EBCDIC Code Page 1147 8-bit French
f8ebcdic297	EBCDIC Code Page 297 8-bit French
hu8abmod	Hungarian 8-bit Special AB Mod
hu8cwi2	Hungarian 8-bit CWI-2
i7dec	DEC VT100 7-bit Italian
i7siemens9780x	Siemens 97801/97808 7-bit Italian
i8ebcdic1144	EBCDIC Code Page 1144 8-bit Italian
i8ebcdic280	EBCDIC Code Page 280/1 8-bit Italian
in8iscii	Multiple-Script Indian Standard 8-bit Latin/Indian
is8macicelandics	Mac Server 8-bit Icelandic
is8macicelandic	Mac Client 8-bit Icelandic
is8pc861	IBM-PC Code Page 861 8-bit Icelandic
iw7is960	Israeli Standard 960 7-bit Latin/Hebrew
iw8ebcdic1086	EBCDIC Code Page 1086 8-bit Hebrew
iw8ebcdic424s	EBCDIC Code Page 424 Server 8-bit Latin/Hebrew
iw8ebcdic424	EBCDIC Code Page 424 8-bit Latin/Hebrew
iw8iso8859p8	ISO 8859-8 Latin/Hebrew
iw8machebrews	Mac Server 8-bit Hebrew
iw8machebrew	Mac Client 8-bit Hebrew

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
iw8mswin1255	MS Windows Code Page 1255 8-bit Latin/Hebrew
iw8pc1507	IBM-PC Code Page 1507/862 8-bit Latin/Hebrew
ja16dbcs	IBM EBCDIC 16-bit Japanese
ja16ebcdic930	IBM DBCS Code Page 290 16-bit Japanese
ja16euutilde	Same as ja16euc except for the way that the wave dash and the tilde are mapped to and from Unicode
ja16euc	EUC 24-bit Japanese
ja16eucyen	EUC 24-bit Japanese with '\' mapped to the Japanese yen character
ja16macsjis	Mac client Shift-JIS 16-bit Japanese
ja16sjistilde	Same as ja16sjis except for the way that the wave dash and the tilde are mapped to and from Unicode.
ja16sjis	Shift-JIS 16-bit Japanese
ja16sjisyen	Shift-JIS 16-bit Japanese with '\' mapped to the Japanese yen character
ja16vms	JVMS 16-bit Japanese
ko16dbcs	IBM EBCDIC 16-bit Korean
ko16ksc5601	KSC5601 16-bit Korean
ko16ksccs	KSCCS 16-bit Korean
ko16mswin949	MS Windows Code Page 949 Korean
la8iso6937	ISO 6937 8-bit Coded Character Set for Text Communication
la8passport	German Government Printer 8-bit All-European Latin
lt8mswin921	MS Windows Code Page 921 8-bit Lithuanian
lt8pc772	IBM-PC Code Page 772 8-bit Lithuanian (Latin/Cyrillic)
lt8pc774	IBM-PC Code Page 774 8-bit Lithuanian (Latin)
lv8pc1117	IBM-PC Code Page 1117 8-bit Latvian
lv8pc8lr	Latvian Version IBM-PC Code Page 866 8-bit Latin/Cyrillic
lv8rst104090	IBM-PC Alternative Code Page 8-bit Latvian (Latin/Cyrillic)
n7siemens9780x	Siemens 97801/97808 7-bit Norwegian
n8pc865	IBM-PC Code Page 865 8-bit Norwegian
ndk7dec	DEC VT100 7-bit Norwegian/Danish
ne8iso8859p10	ISO 8859-10 North European
nee8iso8859p4	ISO 8859-4 North and North-East European
nl7dec	DEC VT100 7-bit Dutch
ru8besta	BESTA 8-bit Latin/Cyrillic
ru8pc855	IBM-PC Code Page 855 8-bit Latin/Cyrillic

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
ru8pc866	IBM-PC Code Page 866 8-bit Latin/Cyrillic
s7dec	DEC VT100 7-bit Swedish
s7siemens9780x	Siemens 97801/97808 7-bit Swedish
s8bs2000	Siemens 9750-62 EBCDIC 8-bit Swedish
s8ebcdic1143	EBCDIC Code Page 1143 8-bit Swedish
s8ebcdic278	EBCDIC Code Page 278/1 8-bit Swedish
se8iso8859p3	ISO 8859-3 South European
sf7ascii	ASCII 7-bit Finnish
sf7dec	DEC VT100 7-bit Finnish
th8macthais	Mac Server 8-bit Latin/Thai
th8macthai	Mac Client 8-bit Latin/Thai
th8tisascii	Thai Industrial Standard 620-2533 - ASCII 8-bit
th8tisebcdics	Thai Industrial Standard 620-2533 - EBCDIC Server 8-bit
th8tisebcdic	Thai Industrial Standard 620-2533 - EBCDIC 8-bit
tr7dec	DEC VT100 7-bit Turkish
tr8dec	DEC 8-bit Turkish
tr8ebcdic1026s	EBCDIC Code Page 1026 Server 8-bit Turkish
tr8ebcdic1026	EBCDIC Code Page 1026 8-bit Turkish
tr8macturkishs	Mac Server 8-bit Turkish
tr8macturkish	Mac Client 8-bit Turkish
tr8mswin1254	MS Windows Code Page 1254 8-bit Turkish
tr8pc857	IBM-PC Code Page 857 8-bit Turkish
us7ascii	ASCII 7-bit American
us8bs2000	Siemens 9750-62 EBCDIC 8-bit American
us8icl	ICL EBCDIC 8-bit American
us8pc437	IBM-PC Code Page 437 8-bit American
vn8mswin1258	MS Windows Code Page 1258 8-bit Vietnamese
vn8vn3	VN3 8-bit Vietnamese
we8bs2000e	Siemens EBCDIC.DF.04-F 8-bit West European with Euro symbol
we8bs200015	Siemens EBCDIC.DF.04-9 8-bit WE & Turkish
we8bs2000	Siemens EBCDIC.DF.04-1 8-bit West European
we8dec	DEC 8-bit West European
we8dg	DG 8-bit West European

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
we8ebcdic1047e	Latin 1/Open Systems 1047
we8ebcdic1047	EBCDIC Code Page 1047 8-bit West European
we8ebcdic1140c	EBCDIC Code Page 1140 Client 8-bit West European
we8ebcdic1140	EBCDIC Code Page 1140 8-bit West European
we8ebcdic1145	EBCDIC Code Page 1145 8-bit West European
we8ebcdic1146	EBCDIC Code Page 1146 8-bit West European
we8ebcdic1148c	EBCDIC Code Page 1148 Client 8-bit West European
we8ebcdic1148	EBCDIC Code Page 1148 8-bit West European
we8ebcdic284	EBCDIC Code Page 284 8-bit Latin American/Spanish
we8ebcdic285	EBCDIC Code Page 285 8-bit West European
we8ebcdic37c	EBCDIC Code Page 37 8-bit Oracle/c
we8ebcdic37	EBCDIC Code Page 37 8-bit West European
we8ebcdic500c	EBCDIC Code Page 500 8-bit Oracle/c
we8ebcdic500	EBCDIC Code Page 500 8-bit West European
we8ebcdic871	EBCDIC Code Page 871 8-bit Icelandic
we8ebcdic924	Latin 9 EBCDIC 924
we8gcos7	Bull EBCDIC GCOS7 8-bit West European
we8hp	HP LaserJet 8-bit West European
we8icl	ICL EBCDIC 8-bit West European
we8iso8859p15	ISO 8859-15 West European
we8iso8859p1	ISO 8859-1 West European
we8iso8859p9	ISO 8859-9 West European & Turkish
we8isoicluk	ICL special version ISO8859-1
we8macroman8s	Mac Server 8-bit Extended Roman8 West European
we8macroman8	Mac Client 8-bit Extended Roman8 West European
we8mswin1252	MS Windows Code Page 1252 8-bit West European
we8ncr4970	NCR 4970 8-bit West European
we8nextstep	NeXTSTEP PostScript 8-bit West European
we8pc850	IBM-PC Code Page 850 8-bit West European
we8pc858	IBM-PC Code Page 858 8-bit West European
we8pc860	IBM-PC Code Page 860 8-bit West European
we8roman8	HP Roman8 8-bit West European
yug7ascii	ASCII 7-bit Yugoslavian

Table A-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
zhs16cgb231280	CGB2312-80 16-bit Simplified Chinese
zhs16dbcs	IBM EBCDIC 16-bit Simplified Chinese
zhs16gbk	GBK 16-bit Simplified Chinese
zhs16maccgb231280	Mac client CGB2312-80 16-bit Simplified Chinese
zht16big5	BIG5 16-bit Traditional Chinese
zht16ccdc	HP CCDC 16-bit Traditional Chinese
zht16dbcs	IBM EBCDIC 16-bit Traditional Chinese
zht16dbt	Taiwan Taxation 16-bit Traditional Chinese
zht16hkscs31	MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (character set conversion to and from Unicode is based on Unicode 3.1)
zht16hkscs	MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (character set conversion to and from Unicode is based on Unicode 3.0)
zht16mswin950	MS Windows Code Page 950 Traditional Chinese
zht32euc	EUC 32-bit Traditional Chinese
zht32sops	SOPS 32-bit Traditional Chinese
zht32tris	TRIS 32-bit Traditional Chinese

Supported Character Sets - Non-Oracle

Identifier to use in parameter files and commands	Character set
UTF-8	ISO-10646 UTF-8, surrogate pairs are 4 bytes per character
UTF-16	ISO-10646 UTF-16
UTF-16BE	UTF-16 Big Endian
UTF-16LE	UTF-16 Little Endian
UTF-32	ISO-10646 UTF-32
UTF-32BE	UTF-32 Big Endian

Identifier to use in parameter files and commands	Character set
UTF-32LE	UTF-32 Little Endian
CESU-8	Similar to UTF-8, correspond to UCS-2 and surrogate pairs are 6 bytes per character
US-ASCII	US-ASCII, ANSI X34-1986
windows-1250	Windows Central Europe
windows-1251	Windows Cyrillic
windows-1252	Windows Latin-1
windows-1253	Windows Greek
windows-1254	Windows Turkish
windows-1255	Windows Hebrew
windows-1256	Windows Arabic
windows-1257	Windows Baltic
windows-1258	Windows Vietnam
windows-874	Windows Thai
cp437	DOS Latin-1
ibm-720	DOS Arabic
cp737	DOS Greek
cp775	DOS Baltic
cp850	DOS multilingual
cp851	DOS Greek-1
cp852	DOS Latin-2
cp855	DOS Cyrillic

Identifier to use in parameter files and commands	Character set
cp856	DOS Cyrillic / IBM
cp857	DOS Turkish
cp858	DOS Multilingual with Euro
cp860	DOS Portuguese
cp861	DOS Icelandic
cp862	DOS Hebrew
cp863	DOS French
cp864	DOS Arabic
cp865	DOS Nordic
cp866	DOS Cyrillic / GOST 19768-87
ibm-867	DOS Hebrew / IBM
cp868	DOS Urdu
cp869	DOS Greek-2
ISO-8859-1	ISO-8859-1 Latin-1/Western Europe
ISO-8859-2	ISO-8859-2 Latin-2/Eastern Europe
ISO-8859-3	ISO-8859-3 Latin-3/South Europe
ISO-8859-4	ISO-8859-4 Latin-4/North Europe
ISO-8859-5	ISO-8859-5 Latin/Cyrillic
ISO-8859-6	ISO-8859-6 Latin/Arabic
ISO-8859-7	ISO-8859-7 Latin/Greek
ISO-8859-8	ISO-8859-8 Latin/Hebrew

Identifier to use in parameter files and commands	Character set
ISO-8859-9	ISO-8859-9 Latin-5/Turkish
ISO-8859-10	ISO-8859-10 Latin-6/Nordic
ISO-8859-11	ISO-8859-11 Latin/Thai
ISO-8859-13	ISO-8859-13 Latin-7/Baltic Rim
ISO-8859-14	ISO-8859-14 Latin-8/Celtic
ISO-8859-15	ISO-8859-15 Latin-9/Western Europe
IBM037	IBM 037-1/697-1 EBCDIC, Brazil, Canada, Netherlands, Portugal, US, and 037/1175 Traditional Chinese
IBM01140	IBM 1140-1/695-1 EBCDIC, Brazil, Canada, Netherlands, Portugal, US, and 1140/1175 Traditional Chinese
IBM273	IBM 273-1/697-1 EBCDIC, Austria, Germany
IBM01141	IBM 1141-1/695-1 EBCDIC, Austria, Germany
IBM277	IBM 277-1/697-1 EBCDIC, Denmark, Norway
IBM01142	IBM 1142-1/695-1 EBCDIC, Denmark, Norway
IBM278	IBM 278-1/697-1 EBCDIC, Finland, Sweden
IBM01143	IBM 1143-1/695-1 EBCDIC, Finland, Sweden
IBM280	IBM 280-1/697-1 EBCDIC, Italy
IBM01144	IBM 1144-1/695-1 EBCDIC, Italy
IBM284	IBM 284-1/697-1 EBCDIC, Latin America, Spain
IBM01145	IBM 1145-1/695-1 EBCDIC, Latin America, Spain
IBM285	IBM 285-1/697-1 EBCDIC, United Kingdom
IBM01146	IBM 1146-1/695-1 EBCDIC, United Kingdom
IBM290	IBM 290 EBCDIC, Japan (Katakana) Extended

**Identifier to use in
parameter files
and commands**

IBM297	IBM 297-1/697-1 EBCDIC, France
IBM01147	IBM 1147-1/695-1 EBCDIC, France
IBM420	IBM 420 EBCDIC, Arabic Bilingual
IBM424	IBM 424/941 EBCDIC, Israel (Hebrew - Bulletin Code)
IBM500	IBM 500-1/697-1 EBCDIC, International
IBM01148	IBM 1148-1/695-1 EBCDIC International
IBM870	IBM 870/959 EBCDIC, Latin-2 Multilingual
IBM871	IBM 871-1/697-1 EBCDIC Iceland
IBM918	IBM EBCDIC code page 918, Arabic 2
IBM1149	IBM 1149-1/695-1, EBCDIC Iceland
IBM1047	IBM 1047/103 EBCDIC, Latin-1 (Open Systems)
ibm-803	IBM 803 EBCDIC, Israel (Hebrew - Old Code)
IBM875	IBM 875 EBCDIC, Greece
ibm-924	IBM 924-1/1353-1 EBCDIC International
ibm-1153	IBM 1153/1375 EBCDIC, Latin-2 Multilingual
ibm-1122	IBM 1122/1037 EBCDIC, Estonia
ibm-1157	IBM 1157/1391 EBCDIC, Estonia
ibm-1112	IBM 1112/1035 EBCDIC, Latvia, Lithuania
ibm-1156	IBM 1156/1393 EBCDIC, Latvia, Lithuania
ibm-4899	IBM EBCDIC code page 4899, Hebrew with Euro
ibm-12712	IBM 12712 EBCDIC, Hebrew (max set including Euro)

Identifier to use in parameter files and commands	Character set
ibm-1097	IBM 1097 EBCDIC, Farsi
ibm-1018	IBM 1018 EBCDIC, Finland Sweden (ISO-7)
ibm-1132	IBM 1132 EBCDIC, Laos
ibm-1137	IBM EBCDIC code page 1137, Devanagari
ibm-1025	IBM 1025/1150 EBCDIC, Cyrillic
ibm-1154	IBM EBCDIC code page 1154, Cyrillic with Euro
IBM1026	IBM 1026/1152 EBCDIC, Latin-5 Turkey
ibm-1155	IBM EBCDIC code page 1155, Turkish with Euro
ibm-1123	IBM 1123 EBCDIC, Ukraine
ibm-1158	IBM EBCDIC code page 1158, Ukrainian with Euro
IBM838	IBM 838/1173 EBCDIC, Thai
ibm-1160	IBM EBCDIC code page 1160, Thai with Euro
ibm-1130	IBM 1130 EBCDIC, Vietnam
ibm-1164	IBM EBCDIC code page 1164, Vietnamese with Euro
ibm-4517	IBM EBCDIC code page 4517, Arabic French
ibm-4971	IBM EBCDIC code page 4971, Greek
ibm-9067	IBM EBCDIC code page 9067, Greek 2005
ibm-16804	IBM EBCDIC code page 16804, Arabic
KOI8-R	Russian and Cyrillic (KOI8-R)
KOI8-U	Ukrainian (KOI8-U)
eucTH	EUC Thai

Identifier to use in parameter files and commands	Character set
ibm-1162	Windows Thai with Euro
DEC-MCS	DEC Multilingual
hp-roman8	HP Latin-1 Roman8
ibm-901	IBM Baltic ISO-8 CCSID 901
ibm-902	IBM Estonia ISO-8 with Euro CCSID 902
ibm-916	IBM ISO8859-8 CCSID
ibm-922	IBM Estonia ISO-8 CCSID 922
ibm-1006	IBM Urdu ISO-8 CCSID 1006
ibm-1098	IBM Farsi PC CCSID 1098
ibm-1124	Ukranian ISO-8 CCSID 1124
ibm-1125	Ukranian without Euro CCSID 1125
ibm-1129	IBM Vietnamese without Euro CCSID 1129
ibm-1131	IBM Belarusi CCSID 1131
ibm-1133	IBM Lao CCSID 1133
ibm-4909	IBM Greek Latin ASCII CCSID 4909
JIS_X201	JIS X201 Japanese
windows-932	Windows Japanese
windows-936	Windows Simplified Chinese
ibm-942	IBM Windows Japanese
windows-949	Windows Korean
windows-950	Windows Traditional Chinese

Identifier to use in parameter files and commands	Character set
eucljis	EUC Japanese
EUC-JP	IBM/MS EUC Japanese
EUC-CN	EUC Simplified Chinese, GBK
EUC-KR	EUC Korean
EUC-TW	EUC Traditional Chinese
ibm-930	IBM 930/5026 Japanese
ibm-933	IBM 933 Korean
ibm-935	IBM 935 Simplified Chinese
ibm-937	IBM 937 Traditional Chinese
ibm-939	IBM 939/5035 Japanese
ibm-1364	IBM 1364 Korean
ibm-1371	IBM 1371 Traditional Chinese
ibm-1388	IBM 1388 Simplified Chinese
ibm-1390	IBM 1390 Japanese
ibm-1399	IBM 1399 Japanese
ibm-5123	IBM CCSID 5123 Japanese
ibm-8482	IBM CCSID 8482 Japanese
ibm-13218	IBM CCSID 13218 Japanese
ibm-16684	IBM CCSID 16684 Japanese
shiftjis	Japanese Shift JIS, Tilde 0x8160 mapped to U+301C
gb18030	GB-18030

Identifier to use in parameter files and commands	Character set
GB2312	GB-2312-1980
GBK	GBK
HZ	HZ GB2312
Ibm-1381	IBM CCSID 1381 Simplified Chinese
Big5	Big5, Traditional Chinese
Big5-HKSCS	Big5, HongKong ext.
Big5-HKSCS2001	Big5, HongKong ext. HKSCS-2001
ibm-950	IBM Big5, CCSID 950
ibm-949	CCSID 949 Korean
ibm-949C	IBM CCSID 949 Korean, has backslash
ibm-971	IBM CCSID 971 Korean EUC, KSC5601 1989
x-IBM1363	IBM CCSID 1363, Korean

B

Supported Locales

This appendix lists the locales that are supported by Oracle GoldenGate. The locale is used when comparing case-insensitive object names.

- af
- af_NA
- af_ZA
- am
- am_ET
- ar
- ar_AE
- ar_BH
- ar_DZ
- ar_EG
- ar_IQ
- ar_JO
- ar_KW
- ar_LB
- ar_LY
- ar_MA
- ar_OM
- ar_QA
- ar_SA
- ar_SD
- ar_SY
- ar_TN
- ar_YE
- as
- as_IN
- az
- az_Cyrl
- az_Cyrl_AZ
- az_Latn
- az_Latn_AZ
- be
- be_BY
- bg
- bg_BG
- bn
- bn_BD
- bn_IN
- ca
- ca_ES

cs
cs_CZ
cy
cy_GB
da
da_DK
de
de_AT
de_BE
de_CH
de_DE
de_LI
de_LU
el
el_CY
el_GR
en
en_AU
en_BE
en_BW
en_BZ
en_CA
en_GB
en_HK
en_IE
en_IN
en_JM
en_MH
en_MT
en_NA
en_NZ
en_PH
en_PK
en_SG
en_TT
en_US
en_US_POSIX
en_VI
en_ZA
en_ZW
eo
es
es_AR
es_BO
es_CL
es_CO
es_CR
es_DO

es_EC
es_ES
es_GT
es_HN
es_MX
es_NI
es_PA
es_PE
es_PR
es_PY
es_SV
es_US
es_UY
es_VE
et
et_EE
eu
eu_ES
fa
fa_AF
fa_IR
fi
fi_FI
fo
fo_FO
fr
fr_BE
fr_CA
fr_CH
fr_FR
fr_LU
fr_MC
ga
ga_IE
gl
gl_ES
gu
gu_IN
gv
gv_GB
haw
haw_US
he
he_IL
hi
hi_IN
hr
hr_HR

hu
hu_HU
hy
hy_AM
hy_AM_REVISED
id
id_ID
is
is_IS
it
it_CH
it_IT
ja
ja_JP
ka
ka_GE
kk
kk_KZ
kl
kl_GL
km
km_KH
kn
kn_IN
ko
ko_KR
kok
kok_IN
kw
kw_GB
lt
lt_LT
lv
lv_LV
mk
mk_MK
ml
ml_IN
mr
mr_IN
ms
ms_BN
ms_MY
mt
mt_MT
nb
nb_NO
nl

nl_BE
nl_NL
nn
nn_NO
om
om_ET
om_KE
or
or_IN
pa
pa_Guru
pa_Guru_IN
pl
pl_PL
ps
ps_AF
pt
pt_BR
pt_PT
ro
ro_RO
ru
ru_RU
ru_UA
sk
sk_SK
sl
sl_SI
so
so_DJ
so_ET
so_KE
so_SO
sq
sq_AL
sr
sr_Cyrl
sr_Cyrl_BA
sr_Cyrl_ME
sr_Cyrl_RS
sr_Latn
sr_Latn_BA
sr_Latn_ME
sr_Latn_RS
sv
sv_FI
sv_SE
sw

sw_KE
sw_TZ
ta
ta_IN
te
te_IN
th
th_TH
ti
ti_ER
ti_ET
tr
tr_TR
uk
uk_UA
ur
ur_IN
ur_PK
uz
uz_Arab
uz_Arab_AF
uz_Cyrl
uz_Cyrl_UZ
uz_Latn
uz_Latn_UZ
vi
vi_VN
zh
zh_Hans
zh_Hans_CN
zh_Hans_SG
zh_Hant
zh_Hant_HK
zh_Hant_MO
zh_Hant_TW

C

About the Oracle GoldenGate Trail

This appendix contains information about the Oracle GoldenGate trail that you may need to know for troubleshooting, for a support case, or for other purposes. To view the Oracle GoldenGate trail records, use the Logdump utility.

Topics:

- [Trail Recovery Mode](#)
- [Trail File Header Record](#)
- [Trail Record Format](#)
- [Example of an Oracle GoldenGate Record](#)
- [Record Header Area](#)
- [Record Data Area](#)
- [Tokens Area](#)
- [Oracle GoldenGate Operation Types](#)
- [Oracle GoldenGate Trail Header Record](#)

Trail Recovery Mode

By default, Extract operates in *append mode*, where if there is a process failure, a recovery marker is written to the trail and Extract appends recovery data to the file so that a history of all prior data is retained for recovery purposes.

In append mode, the Extract initialization determines the identity of the last complete transaction that was written to the trail at startup time. With that information, Extract ends recovery when the commit record for that transaction is encountered in the data source; then it begins new data capture with the next committed transaction that qualifies for extraction and begins appending the new data to the trail. A data pump or Replicat starts reading again from that recovery point.

Overwrite mode is another version of Extract recovery that was used in versions of Oracle GoldenGate prior to version 10.0. In these versions, Extract overwrites the existing transaction data in the trail after the last write-checkpoint position, instead of appending the new data. The first transaction that is written is the first one that qualifies for extraction after the last read checkpoint position in the data source.

If the version of Oracle GoldenGate on the target is older than version 10, Extract will automatically revert to overwrite mode to support backward compatibility. This behavior can be controlled manually with the `RECOVERYOPTIONS` parameter.

Trail File Header Record

As of Oracle GoldenGate version 10.0, each file of a trail contains a *file header record* that is stored at the beginning of the file. The file header contains information about the trail file itself. Previous versions of Oracle GoldenGate do not contain this header.

Because all of the Oracle GoldenGate processes are decoupled and thus can be of different Oracle GoldenGate versions, the file header of each trail file contains a version indicator. By default, the version of a trail file is the current version of the process that created the file. If you need to set the version of a trail, use the `FORMAT` option of the `EXTTRAIL`, `EXTFILE`, `RMTTRAIL`, or `RMTFILE` parameter.

To ensure forward and backward compatibility of files among different Oracle GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer Oracle GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is `COMPATIBILITY` and can be viewed in the Logdump utility and also by retrieving it with the `GGFILEHEADER` option of the `@GETENV` function.

A trail or extract file must have a version that is equal to, or lower than, that of the process that *reads* it. Otherwise the process will abend. Additionally, Oracle GoldenGate forces the output trail or file of a data pump to be the same version as that of its input trail or file. Upon restart, Extract rolls a trail to a new file to ensure that each file is of only one version (unless the file is empty).

Trail Record Format

Each change record written by Oracle GoldenGate to a trail or extract file includes a header area, a data area, and possibly a user token area. The record header contains information about the transaction environment, and the data area contains the actual data values that were extracted. The token area contains information that is specified by Oracle GoldenGate users for use in column mapping and conversion.

Oracle GoldenGate trail files are unstructured. You can view Oracle GoldenGate records with the Logdump utility provided with the Oracle GoldenGate software. For more information, see Logdump Reference for Oracle GoldenGate.

Note:

As enhancements are made to the Oracle GoldenGate software, the trail record format is subject to changes that may not be reflected in this documentation. To view the current structure, use the Logdump utility.

Example of an Oracle GoldenGate Record

The following illustrates an Oracle GoldenGate record as viewed with Logdump. The first portion (the list of fields) is the header and the second portion is the data area. The record looks similar to this on all platforms supported by Oracle GoldenGate.

```

Commands to show headers, column detail,
and user tokens, and to go to next record
Logdump 59 >open c:\goldengate802\dirat\cc000000
Current LogTrail is c:\goldengate802\dirat\cc000000
Logdump 60 >hdr on
Logdump 61 >detail on
Logdump 62 >detail data
Logdump 63 >usertoken on
Logdump 64 >n
Header area: contains transaction information
-----
Hdr-Ind      : E <x45>      Partition   :      <x04>
UndoFlag    :      <x00>      BeforeAfter:  a  <x41>
RecLength   : 64 <x0040>    IO Time      : 2011/01/24 14:45:26.000.000
IOType      : 5 <x05>      OrigNode    : 255 <x0f>
TransInd    :      <x03>      FormatType  : R <x52>
SysLen     : 0 <x00>      Incomplete  :      <x00>
AuditRBA    : 41          AuditPos    : 92002584
Continued   : N <x00>      RecCount    : 1 <x01>
-----
Operation type and time record was written
2011/01/24 14:45:26.000.000 Insert Len 64 RBA 0
Source object Name: DDIEC.DEPARTMENTS
After Image:
Image type: could be a before/after
0000 4164 6069 6069 7274 7261 7469 676E 0002 0000 : .....Administration..
0000 0000 0000 0000 00C8 0003 000A 0000 0000 0000 : .....
0000 06A4 0 <x0000> Len 10 <x000a> : .....
Column information with data, or could be sequence information
Column 1 <x0001> Len 18 <x0012> : .....
0000 000E 4164 6069 6069 7374 7261 7469 6F6E : .....Administration
Column 2 <x0002> Len 10 <x000a> : .....
0000 0000 0000 0000 00C8 : .....
Column 3 <x0003> Len 10 <x000a> : .....
0000 0000 0000 0000 06A4 : .....
User token area
User tokens: 7 bytes
5465 7374 0031 00 : Test.1.
-----
Record data, in hex format
Length of record
RBA position of record in the trail file
Record data, in ASCII format

```

Record Header Area

The Oracle GoldenGate record header provides metadata of the data that is contained in the record and includes the following information.

- The operation type, such as an insert, update, or delete
- The before or after indicator for updates
- Transaction information, such as the transaction group and commit timestamp
- [Description of Header Fields](#)
- [Using Header Data](#)

Description of Header Fields

The following describes the fields of the Oracle GoldenGate record header. Some fields apply only to certain platforms.

Table C-1 Oracle GoldenGate record header fields

Field	Description
Hdr-Ind	Should always be a value of E, indicating that the record was created by the Extract process. Any other value indicates invalid data.
UndoFlag	(NonStop) Conditionally set if Oracle GoldenGate is extracting aborted transactions from the TMF audit trail. Normally, UndoFlag is set to zero, but if the record is the backout of a previously successful operation, then UndoFlag will be set to 1. An undo that is performed by the disc process because of a constraint violation is not marked as an undo.
RecLength	The length, in bytes, of the record buffer.
IOType	The type of operation represented by the record. See Table C-2 for a list of operation types.

Table C-1 (Cont.) Oracle GoldenGate record header fields

Field	Description
TransInD	The place of the record within the current transaction. Values are: 0 — first record in transaction 1 — neither first nor last record in transaction 2 — last record in the transaction 3 — only record in the transaction
SyskeyLen	(NonStop) The length of the system key (4 or 8 bytes) if the source is a NonStop file and has a system key. If a system key exists, the first SyskeyLen bytes of the record are the system key. Otherwise, SyskeyLen is 0.
AuditRBA	Identifies the transaction log identifier, such as the Oracle redo log sequence number.
Continued	(Windows and UNIX) Identifies whether or not the record is a segment of a larger piece of data that is too large to fit within one record. LOBs, CLOBs, and some VARCHARs are stored in segments. Unified records that contain both before and after images in a single record (due to the UPDATERECORDFORMAT parameter) may exceed the maximum length of a record and may also generate segments. Y — the record is a segment; indicates to Oracle GoldenGate that this data continues to another record. N — there is no continuation of data to another segment; could be the last in a series or a record that is not a segment of larger data.
Partition	For Windows and UNIX records, this field will always be a value of 4 (FieldComp compressed record in internal format). For these platforms, the term Partition does not indicate that the data represents any particular logical or physical partition within the database structure. For NonStop records, the value of this field depends on the record type: <ul style="list-style-type: none"> In the case of BulkIO operations, Partition indicates the number of the source partition on which the bulk operation was performed. It tells Oracle GoldenGate which source partition the data was originally written to. Replicat uses the Partition field to determine the name of the target partition. The file name in the record header will always be the name of the primary partition. Valid values for BulkIO records are 0 through 15. For other non-bulk NonStop operations, the value can be either 0 or 4. A value of 4 indicates that the data is in FieldComp record format.
BeforeAfter	Identifies whether the record is a before (B) or after (A) image of an update operation. Records that combine both before and after images as the result of the UPDATERECORDFORMAT parameter are marked as after images. Inserts are always after images, deletes are always before images.
IO Time	The time when the operation occurred, in local time of the source system, in GMT format. This time may be the same or different for every operation in a transaction depending on when the operation occurred.

Table C-1 (Cont.) Oracle GoldenGate record header fields

Field	Description
OrigNode	(NonStop) The node number of the system where the data was extracted. Each system in a NonStop cluster has a unique node number. Node numbers can range from 0 through 255. For records other than NonStop in origin, OrigNode is 0.
FormatType	Identifies whether the data was read from the transaction log or fetched from the database. F — fetched from database R — readable in transaction log
Incomplete	This field is obsolete.
AuditPos	Identifies the position in the transaction log of the data.
RecCount	(Windows and UNIX) Used for LOB data when it must be split into chunks to be written to the Oracle GoldenGate file. RecCount is used to reassemble the chunks.

Using Header Data

Some of the data available in the Oracle GoldenGate record header can be used for mapping by using the `GGHEADER` option of the `@GETENV` function or by using any of the following transaction elements as the source expression in a `COLMAP` statement in the `TABLE` or `MAP` parameter.

- `GG$TRANS_TIMESTAMP`
- `GG$TRANS_RBA`
- `GG$OP_TYPE`
- `GG$BEFORE_AFTER_IND`

Record Data Area

The data area of the Oracle GoldenGate trail record contains the following:

- The time that the change was written to the Oracle GoldenGate file
- The type of database operation
- The length of the record
- The relative byte address within the trail file
- The table name
- The data changes in hex format

The following explains the differences in record image formats used by Oracle GoldenGate on Windows, UNIX, Linux, and NonStop systems.

- [Full Record Image Format \(NonStop Sources\)](#)
- [Compressed Record Image Format \(Windows, UNIX, Linux Sources\)](#)

Full Record Image Format (NonStop Sources)

A *full record image* contains the values of all of the columns of a processed row. Full record image format is generated in the trail when the source system is HP NonStop, and only when the `IOType` specified in the record header is one of the following:

3 – Delete
5 – Insert
10 – Update

Each full record image has the same format as if retrieved from a program reading the original file or table directly. For SQL tables, datetime fields, nulls, and other data is written exactly as a program would select it into an application buffer. Although datetime fields are represented internally as an eight-byte timestamp, their external form can be up to 26 bytes expressed as a string. Enscribe records are retrieved as they exist in the original file.

When the operation type is `Insert` or `Update`, the image contains the contents of the record *after* the operation (the *after image*). When the operation type is `Delete`, the image contains the contents of the record *before* the operation (the *before image*).

For records generated from an Enscribe database, full record images are output unless the original file has the `AUDITCOMPRESS` attribute set to `ON`. When `AUDITCOMPRESS` is `ON`, compressed update records are generated whenever the original file receives an update operation. (A full image can be retrieved by the Extract process by using the `FETCHCOMPS` parameter.)

Compressed Record Image Format (Windows, UNIX, Linux Sources)

A *compressed record image* contains only the key (primary, unique, `KEYCOLS`) and the columns that changed in the processed row. By default, trail records written by processes on Windows and UNIX systems are always compressed. The format of a compressed record is as follows:

```
column_index column_length column_data[...]
```

Where:

- `column_index` is the ordinal index of the column within the source table (2 bytes).
- `column_length` is the length of the data (2 bytes).
- `column_data` is the data, including `NULL` or `VARCHAR` length indicators.

Enscribe records written from the NonStop platform may be compressed. The format of a compressed Enscribe record is as follows:

```
field_offset field_length field_value[...]
```

Where:

- `field_offset` is the offset within the original record of the changed value (2 bytes).
- `field_length` is the length of the data (2 bytes).
- `field_value` is the data, including `NULL` or `VARCHAR` length indicators.

The first field in a compressed Enscribe record is the primary or system key.

Tokens Area

The trail record also can contain two areas for tokens. One is for internal use and is not documented here, and the other is the user tokens area. User tokens are environment values that are captured and stored in the trail record for replication to target columns or other purposes. If used, these tokens follow the data portion of the record and appear similar to the following when viewed with Logdump:

Parameter	Value
TKN-HOST	: syshq
TKN-GROUP	: EXTORA
TKN-BA_IND	: AFTER
TKN-COMMIT_TS	: 2011-01-24 17:08:59.000000
TKN-POS	: 3604496
TKN-RBA	: 4058
TKN-TABLE	: SOURCE.CUSTOMER
TKN-OPTYPE	: INSERT
TKN-LENGTH	: 57
TKN-TRAN_IND	: BEGIN

Oracle GoldenGate Operation Types

The following are some of the Oracle GoldenGate operation types. Types may be added as new functionality is added to Oracle GoldenGate. For a more updated list, use the `SHOW RECTYPE` command in the Logdump utility.

Table C-2 Oracle GoldenGate Operation Types

Type	Description	Platform
1-Abort	A transaction aborted.	NSK TMF
2-Commit	A transaction committed.	NSK TMF
3-Delete	A record/row was deleted. A Delete record usually contains a full record image. However, if the <code>COMPRESSEDELETES</code> parameter was used, then only key columns will be present.	All
4-EndRollback	A database rollback ended	NSK TMF
5-Insert	A record/row was inserted. An Insert record contains a full record image.	All
6-Prepared	A networked transaction has been prepared to commit.	NSK TMF
7-TMF-Shutdown	A TMF shutdown occurred.	NSK TMF
8-TransBegin	No longer used.	NSK TMF
9-TransRelease	No longer used.	NSK TMF

Table C-2 (Cont.) Oracle GoldenGate Operation Types

Type	Description	Platform
10-Update	A record/row was updated. An Update record contains a full record image. Note: If the partition indicator in the record header is 4, then the record is in FieldComp format (see below) and the update is compressed.	All
11-UpdateComp	A record/row in TMF AuditComp format was updated. In this format, only the changed bytes are present. A 4-byte descriptor in the format of 2-byte_offset2-byte_length precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data.	NSK TMF
12-FileAlter	An attribute of a database file was altered.	NSK
13-FileCreate	A database file was created.	NSK
14-FilePurge	A database file was deleted.	NSK
15-FieldComp	A row in a SQL table was updated. In this format, only the changed bytes are present. Before images of unchanged columns are not logged by the database. A 4-byte descriptor in the format of 2-byte_offset2-byte_length precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data. A partition indicator of 4 in the record header indicates FieldComp format.	All
16-FileRename	A file was renamed.	NSK
17-AuxPointer	Contains information about which AUX trails have new data and the location at which to read.	NSK TMF
18-NetworkCommit	A networked transaction committed.	NSK TMF
19-NetworkAbort	A networked transaction was aborted.	NSK TMF
90-(GGS)SQLCol	A column or columns in a SQL table were added, or an attribute changed.	NSK
100-(GGS)Purgedata	All data was removed from the file (PURGEDATA).	NSK
101-(GGS)Purge(File)	A file was purged.	NSK non-TMF
102-(GGS)Create(File)	A file was created. The Oracle GoldenGate record contains the file attributes.	NSK non-TMF
103-(GGS)Alter(File)	A file was altered. The Oracle GoldenGate record contains the altered file attributes.	NSK non-TMF
104-(GGS)Rename(File)	A file was renamed. The Oracle GoldenGate record contains the original and new names.	NSK non-TMF
105-(GGS)Setmode	A SETMODE operation was performed. The Oracle GoldenGate record contains the SETMODE information.	NSK non-TMF
106-GGSChangeLabel	A CHANGELABEL operation was performed. The Oracle GoldenGate record contains the CHANGELABEL information.	NSK non-TMF

Table C-2 (Cont.) Oracle GoldenGate Operation Types

Type	Description	Platform
107-(GGS)Control	A CONTROL operation was performed. The Oracle GoldenGate record contains the CONTROL information.	NSK non-TMF
115 and 117 (GGS)KeyFieldComp(32)	A primary key was updated. The Oracle GoldenGate record contains the before image of the key and the after image of the key and the row. The data is in FieldComp format (compressed), meaning that before images of unchanged columns are not logged by the database.	Windows and UNIX
116-LargeObject 116-LOB	Identifies a RAW, BLOB, CLOB, or LOB column. Data of this type is stored across multiple records.	Windows and UNIX
132-(GGS)SequenceOp	Identifies an operation on a sequence.	Windows and UNIX
134-UNIFIED UPDATE 135-UNIFIED PKUPDATE	Identifies a unified trail record that contains both before and after values in the same record. The before image in a UNIFIED UPDATE contains all of the columns that are available in the transaction record for both the before and after images. The before image in a UNIFIED PKUPDATE contains all of the columns that are available in the transaction record, but the after image is limited to the primary key columns and the columns that were modified in the UPDATE.	Windows and UNIX
160 - DDL_Op	Identifies a DDL operation	Windows and UNIX
161-RecordFragment	Identifies part of a large row that must be stored across multiple records (more than just the base record).	Windows and UNIX
200-GGSUnstructured Block 200-BulkIO	A BULKIO operation was performed. The Oracle GoldenGate record contains the RAW DP2 block.	NSK non-TMF
201 through 204	These are different types of NonStop trace records. Trace records are used by Oracle GoldenGate support analysts. The following are descriptions. <ul style="list-style-type: none"> ARTYPE_FILECLOSE_GGS 201 — the source application closed a file that was open for unstructured I/O. Used by Replicat ARTYPE_LOGGERTS_GGS 202 — Logger heartbeat record ARTYPE_EXTRACTERTS_GGS 203 — unused ARTYPE_COLLECTORTS_GGS 204 — unused 	NSK non-TMF
205-GGSComment	Indicates a comment record created by the Logdump utility. Comment records are created by Logdump at the beginning and end of data that is saved to a file with Logdump's SAVE command.	All

Table C-2 (Cont.) Oracle GoldenGate Operation Types

Type	Description	Platform
249 through 254	<p>These are different types of NonStop trace records. Trace records are used by Oracle GoldenGate support analysts. The following are descriptions.</p> <ul style="list-style-type: none"> ARTYPE_LOGGER_ADDED_STATS 249 — a stats record created by Logger when the source application closes its open on Logger (if SENDERSTATS is enabled and stats are written to the logtrail) ARTYPE_LIBRARY_OPEN 250 — written by BASELIB to show that the application opened a file ARTYPE_LIBRARY_CLOSE 251 — written by BASELIB to show that the application closed a file. ARTYPE_LOGGER_ADDED_OPEN 252 — unused ARTYPE_LOGGER_ADDED_CLOSE 253 — unused ARTYPE_LOGGER_ADDED_INFO 254 — written by Logger and contains information about the source application that performed the I/O in the subsequent record (if SENDERSTATS is enabled and stats are written to the logtrail). The file name in the trace record is the object file of the application. The trace data has the application process name and the name of the library (if any) that it was running with. 	NSK non-TMF

Oracle GoldenGate Trail Header Record

In addition to the transaction-related records that are in the Oracle GoldenGate trail, each trail file contains a file header.

The file header is stored as a record at the beginning of a trail file preceding the data records. The information that is stored in the trail header provides enough information about the records to enable an Oracle GoldenGate process to determine whether the records are in a format that the current version of Oracle GoldenGate supports.

The trail header fields are stored as tokens, where the token format remains the same across all versions of Oracle GoldenGate. If a version of Oracle GoldenGate does not support any given token, that token is ignored. Deprecated tokens are assigned a default value to preserve compatibility with previous versions of Oracle GoldenGate.

You can view the trail header with the `FILEHEADER` command in the Logdump utility. For more information about the tokens in the file header, see *Logdump Reference for Oracle GoldenGate*.

D

Using the Commit Sequence Number

This appendix contains information about using the Oracle GoldenGate Commit Sequence Number (CSN) with Oracle and non-Oracle databases.

All database platforms except Oracle, DB2 LUW, and DB2 z/OS have fixed-length CSNs, which are padded with leading zeroes as required to fill the fixed length. CSNs that contain multiple fields can be padded within each field. For more information on CSN, see Overview of CSN in *Understanding Oracle GoldenGate*

MySQL does not create a transaction ID as part of its event data, so Oracle GoldenGate considers a unique transaction identifier to be a *combination* of the following:

- the log file number of the log file that contains the `START TRANSACTION` record for the transaction that is being identified
- the record offset of that record

Table D-1 Oracle GoldenGate CSN Values Per Database

Database	CSN Value
DB2 for i	<i>sequence_number</i> Where: <ul style="list-style-type: none">• <i>sequence_number</i> is the fixed-length, 20 digit, decimal-based DB2 for i system sequence number. Example: 12345678901234567890
DB2 LUW	LRI Where: For version 10.1 and later, <i>LRI</i> is a period-separated pair of numbers for the DB2 log record identifier. Example: 123455.34645
DB2 z/OS	<i>RBA</i> where: <ul style="list-style-type: none">• <i>RBA</i> is the 6-byte relative byte address of the commit record within the transaction log. Example: 1274565892

Table D-1 (Cont.) Oracle GoldenGate CSN Values Per Database

Database	CSN Value
MySQL	<p><i>LogNum:LogPosition</i></p> <p>Where:</p> <ul style="list-style-type: none"> <i>LogNum</i> is the the name of the log file that contains the START TRANSACTION record for the transaction that is being identified. <i>LogPosition</i> is the event offset value of that record. Event offset values are stored in the record header section of a log record. <p>For example, if the log number is 12 and the log position is 121, the CSN is:</p> <p>000012:000000000000121</p>
Oracle	<p><i>system_change_number</i></p> <p>Where:</p> <ul style="list-style-type: none"> <i>system_change number</i> is the Oracle SCN value. <p>Example:</p> <p>6488359</p>
SQL Server	<p>Can be any of these, depending on how the database returns it:</p> <ul style="list-style-type: none"> Colon separated hex string (8:8:4) padded with leading zeroes and 0X prefix Colon separated decimal string (10:10:5) padded with leading zeroes Colon separated hex string with 0X prefix and without leading zeroes Colon separated decimal string without leading zeroes Decimal string <p>Where:</p> <ul style="list-style-type: none"> The first value is the virtual log file number, the second is the segment number within the virtual log, and the third is the entry number. <p>Examples:</p> <p>0X00000d7e:0000036b:01bd 0000003454:0000000875:00445 0Xd7e:36b:1bd 3454:875:445 3454000000087500445</p>
Teradata	<p><i>sequence_ID</i></p> <p>Where:</p> <ul style="list-style-type: none"> <i>sequence_ID</i> is a generic fixed-length printable sequence ID. <p>Example:</p> <p>0x080000000000000D700000021</p>

E

About Checkpoints

This appendix provides information about checkpoints. When working with Oracle GoldenGate, you might need to refer to the checkpoints that are made by a process. Checkpoints save the state of the process for recovery purposes. Extract and Replicat use checkpoints.

Topics:

- [Extract Checkpoints](#)
- [Replicat Checkpoints](#)
- [Internal Checkpoint Information](#)
- [Oracle GoldenGate Checkpoint Tables](#)

Extract Checkpoints

Extract checkpoint positions are composed of read checkpoints in the data source and write checkpoints in the trail. The following is a sampling of checkpoint information displayed with the `INFO EXTRACT` command with the `SHOWCH` option. In this case, the data source is an Oracle RAC database cluster, so there is thread information included in the output. You can view past checkpoints by specifying the number of them that you want to view after the `SHOWCH` argument.

Example E-1 INFO EXTRACT with SHOWCH

```
EXTRACT    JC108XT Last Started 2011-01-01 14:15   Status ABENDED
Checkpoint Lag      00:00:00 (updated 00:00:01 ago)
Log Read Checkpoint File /orarc/oradata/racq/redo01.log
                  2011-01-01 14:16:45 Thread 1, Seqno 47, RBA 68748800
Log Read Checkpoint File /orarc/oradata/racq/redo04.log
                  2011-01-01 14:16:19 Thread 2, Seqno 24, RBA 65657408
```

Current Checkpoint Detail:

Read Checkpoint #1

Oracle RAC Redo Log

Startup Checkpoint (starting position in data source):

```
Thread #: 1
Sequence #: 47
RBA: 68548112
Timestamp: 2011-01-01 13:37:51.000000
SCN: 0.8439720
Redo File: /orarc/oradata/racq/redo01.log
```

Recovery Checkpoint (position of oldest unprocessed transaction in data source):

```
Thread #: 1
Sequence #: 47
RBA: 68748304
Timestamp: 2011-01-01 14:16:45.000000
SCN: 0.8440969
```

Redo File: /orarc/oradata/racq/redo01.log

Current Checkpoint (position of last record read in the data source):

Thread #: 1
Sequence #: 47
RBA: 68748800
Timestamp: 2011-01-01 14:16:45.000000
SCN: 0.8440969
Redo File: /orarc/oradata/racq/redo01.log

Read Checkpoint #2

Oracle RAC Redo Log

Startup Checkpoint(starting position in data source):

Sequence #: 24
RBA: 60607504
Timestamp: 2011-01-01 13:37:50.000000
SCN: 0.8439719
Redo File: /orarc/oradata/racq/redo04.log

Recovery Checkpoint (position of oldest unprocessed transaction in data source):

Thread #: 2
Sequence #: 24
RBA: 65657408
Timestamp: 2011-01-01 14:16:19.000000
SCN: 0.8440613
Redo File: /orarc/oradata/racq/redo04.log

Current Checkpoint (position of last record read in the data source):

Thread #: 2
Sequence #: 24
RBA: 65657408
Timestamp: 2011-01-01 14:16:19.000000
SCN: 0.8440613
Redo File: /orarc/oradata/racq/redo04.log

Write Checkpoint #1

GGs Log Trail

Current Checkpoint (current write position):

Sequence #: 2
RBA: 2142224
Timestamp: 2011-01-01 14:16:50.567638
Extract Trail: ./dirdat/eh

Header:

Version = 2
Record Source = A
Type = 6
Input Checkpoints = 2
Output Checkpoints = 1

File Information:

Block Size = 2048
Max Blocks = 100
Record Length = 2048
Current Offset = 0

```
Configuration:
  Data Source = 3
  Transaction Integrity = 1
  Task Type = 0

Status:
  Start Time = 2011-01-01 14:15:14
  Last Update Time = 2011-01-01 14:16:50
  Stop Status = A
  Last Result = 400
```

See [Internal Checkpoint Information](#) for information about the internal information that starts with the `Header` entry in the `SHOWCH` output.

- [About Extract read checkpoints](#)
- [About Extract Write Checkpoints](#)

About Extract read checkpoints

Extract places read checkpoints in the data source.

- [Startup Checkpoint](#)
- [Recovery Checkpoint](#)
- [Current Checkpoint](#)

Startup Checkpoint

The startup checkpoint is the first checkpoint that is made in the data source when the process starts. This statistic is composed of the following:

- `Thread #`: The number of the Extract thread that made the checkpoint, if Oracle GoldenGate is running in an Oracle RAC environment. Otherwise, this statistic is not displayed.
- `Sequence #`: The sequence number of the transaction log where the checkpoint was made.
- `RBA`: The relative byte address of the record at which the checkpoint was made.
- `Timestamp`: The timestamp of the record at which the checkpoint was made.
- `SCN`: The system change number of the record at which the checkpoint was made.
- `Redo File`: The path name of the transaction log containing the record where the checkpoint was made.

Recovery Checkpoint

The recovery checkpoint is the position in the data source of the record containing the oldest transaction not yet processed by Extract. The fields for this statistic are the same as those of the other read checkpoint types.

Current Checkpoint

The current checkpoint is the position of the last record read by Extract in the data source. This should match the `Log Read Checkpoint` statistic shown in the summary

and in the basic `INFO EXTRACT` command without options. The fields for this statistic are the same as those of the other read checkpoint types.

About Extract Write Checkpoints

Extract places a write checkpoint, known as the current checkpoint, in the trail. The current checkpoint is the position in the trail where Extract is currently writing. This statistic is composed of the following:

- **Sequence #:** The sequence number of the trail file where the checkpoint was written.
- **RBA:** The relative byte address of the record in the trail file at which the checkpoint was made.
- **Timestamp:** The timestamp of the record at which the checkpoint was made.
- **Extract trail:** The relative path name of the trail.
- **Trail Type:** Identifies the trail type. `EXTTRAIL` identifies the trail as a local trail, which means that it is directly accessible by Oracle GoldenGate processes through the host filesystem. `RMTTRAIL` identifies the trail as a remote trail, which means it is not directly accessible by Oracle GoldenGate processes through the host filesystem. A trail stored on a shared network device and accessible through NFS-like services are considered local because they are accessible transparently through the host filesystem.

Replicat Checkpoints

Replicat makes checkpoints in the trail file to mark its last read position. To view process checkpoints, use the `INFO REPLICAT` command with the `SHOWCH` option. The basic command shows current checkpoints. To view a specific number of previous checkpoints, type the value after the `SHOWCH` argument.

Example E-2 INFO REPLICAT, SHOWCH

```
REPLICAT   JC108RP   Last Started 2011-01-12 13:10   Status RUNNING
Checkpoint Lag      00:00:00 (updated 111:46:54 ago)
Log Read Checkpoint File ./dirdat/eh000000000
                  First Record RBA 3702915
Current Checkpoint Detail:
  Read Checkpoint #1
  GGS Log Trail
  Startup Checkpoint(starting position in data source):
  Sequence #: 0
  RBA: 3702915
  Timestamp: Not Available
  Extract Trail: ./dirdat/eh
  Current Checkpoint (position of last record read in the data source):
  Sequence #: 0
  RBA: 3702915
  Timestamp: Not Available
  Extract Trail: ./dirdat/eh
Header:
Version = 2
Record Source = A
Type = 1
# Input Checkpoints = 1
# Output Checkpoints = 0
```

```
File Information:
Block Size = 2048
Max Blocks = 100
Record Length = 2048
Current Offset = 0
Configuration:
Data Source = 0
Transaction Integrity = -1
Task Type = 0
Status:
Start Time = 2011-01-12 13:10:13
Last Update Time = 2011-01-12 21:23:31
Stop Status = A
Last Result = 400
```

See [Internal Checkpoint Information](#) for information about the internal information that starts with the `Header` entry in the `SHOWCH` output.

- [About Replicat Checkpoints](#)

About Replicat Checkpoints

The following describes the detail of the Replicat checkpoints in the trail.

- [Startup Checkpoint](#)
- [Current Checkpoint](#)

Startup Checkpoint

The *startup checkpoint* is the first checkpoint made in the trail when the process starts. Comprising this statistic are:

- `Sequence #`: The sequence number of the trail file where the checkpoint was written.
- `RBA`: The relative byte address of the record at which the checkpoint was made.
- `Timestamp`: The timestamp of the record at which the checkpoint was made.
- `Extract Trail`: The relative path name of the trail.

Current Checkpoint

The *current checkpoint* is the position of the last record read by Replicat in the trail. This should match the `Log Read Checkpoint` statistic shown in the summary and in the basic `INFO REPLICAT` command without options. The fields for this statistic are the same as those of the `Startup Checkpoint`.

Internal Checkpoint Information

The `INFO` command with the `SHOWCH` option not only displays current checkpoint entries, but it also displays metadata information about the record itself. This information is not documented and is for use by the Oracle GoldenGate processes and by support personnel when resolving a support case. The metadata is contained in the following entries in the `SHOWCH` output.

```
Header:
Version = 2
Record Source = A
Type = 1
# Input Checkpoints = 1
# Output Checkpoints = 0
File Information:
Block Size = 2048
Max Blocks = 100
Record Length = 2048
Current Offset = 0
Configuration:
Data Source = 0
Transaction Integrity = -1
Task Type = 0
Status:
Start Time = 2011-01-12 13:10:13
Last Update Time = 2011-01-12 21:23:31
Stop Status = A
Last Result = 400
```

Oracle GoldenGate Checkpoint Tables

When database checkpoints are being used, Oracle GoldenGate creates a checkpoint table with a user-defined name in the database upon execution of the `ADD CHECKPOINTTABLE` command, or a user can create the table by using the `chkpt_db_create.sql` script (where *db* is an abbreviation of the type of database that the script supports).

There are two tables: the main checkpoint table and an auxiliary checkpoint table that is created automatically. The auxiliary table, known as the *transaction table*, bears the name of the primary checkpoint table appended with `_lox`. Each Replicat, or each thread of a coordinated Replicat, uses one row in the checkpoint table to store its progress information.

At checkpoint time, there typically are some number of transactions (among the total *n* transactions) that were applied, and the rest are still in process. For example, if Replicat is processing a group of *n* transactions ranging from CSN1 to CSN3. CSN1 is the high watermark and CSN3 is the low watermark. Any transaction with a CSN higher than the high watermark has not been processed, and any transaction with a CSN lower than the low watermark has already been processed. Completed transactions are stored in the `LOG_CMPLT_XID` column of the checkpoint table. Any overflow of these transactions is stored in the transaction table (auxiliary checkpoint table) in the `LOG_CMPLT_XID` column of that table.

Currently, Replicat (or each Replicat thread of a coordinated Replicat) applies transactions serially (not in parallel); therefore, the high watermark (the `LOG_CSN` value in the table) is always the same as the low watermark (the `LOG_CMPLT_CSN` value in the table), and there typically is only one transaction ID in the `LOG_CMPLT_XID` column. The only exception is when there are multiple transactions sharing the same CSN.

Do not change the names or attributes of the columns in these tables. You can change table storage attributes as needed.

Table E-1 Checkpoint table definition

Column	Description
GROUP_NAME (primary key)	The name of a Replicat group using this table for checkpoints. There can be multiple Replicat groups using the same table. This column is part of the primary key.
GROUP_KEY (primary key)	A unique identifier that, together with GROUPNAME, uniquely identifies a checkpoint regardless of how many Replicat groups are writing to the same table. This column is part of the primary key.
SEQNO	The sequence number of the input trail that Replicat was reading at the time of the checkpoint.
RBA	The relative byte address that Replicat reached in the trail identified by SEQNO. RBA + SEQNO provide an absolute position in the trail that identifies the progress of Replicat at the time of checkpoint.
AUDIT_TS	The timestamp of the commit of the source transaction.
CREATE_TS	The date and time when the checkpoint table was created.
LAST_UPDATE_TS	The date and time when the checkpoint table was last updated.
CURRENT_DIR	The current Oracle GoldenGate home directory or folder.
LOG_CSN	Stores the high watermark, or the upper boundary, of the CSNs. Any transaction with a CSN higher than this value has not been processed.
LOG_XID	Not used. Retained for backward compatibility.
LOG_CMPLT_CSN	Stores the low watermark, or the lower boundary, of the CSNs. Any transaction with a lower CSN than this value has already been processed.
LOG_CMPLT_XIDS	Stores the transactions between the high and low watermarks that are already applied.
VERSION	The version of the checkpoint table format. Enables future enhancements to be identified as version numbers of the table.

Table E-2 Transaction table definition

Column	Description
GROUP_NAME	The name of a Replicat group using this table for checkpoints. There can be multiple Replicat groups using the same table. This column is part of the primary key of the transaction table.
GROUP_KEY	A unique identifier that, together with GROUPNAME, uniquely identifies a checkpoint regardless of how many Replicat groups are writing to the same table. This column is part of the primary key of the transaction table.
LOG_CMPLT_CSN	The foreign key that references the checkpoint table. This column is part of the primary key of the transaction table.
LOG_CMPLT_XIDS_SEQ	Creates unique rows in the event there are so many overflow transactions that multiple rows are required to store them all. This column is part of the primary key of the transaction table.

Table E-2 (Cont.) Transaction table definition

Column	Description
LOG_CMPLT_XIDS	Stores the overflow of transactions between the high and low watermarks that are already applied.