Oracle® GoldenGate

Windows and UNIX Administrator's Guide

11*g* Release 2 (11.2.1.0.0)

E27273-01

February 2012

Oracle GoldenGate Windows and UNIX Administrator's Guide 11g Release 2 (11.2.1.0.0)

E27273-01

Contents

Preface **About the Oracle GoldenGate guides** 6
Typographic conventions used in this manual 7
Getting more help with Oracle GoldenGate 7
Chapter 1 **Introduction to Oracle GoldenGate** 9
Oracle GoldenGate supported processing methods and databases 9
Overview of the Oracle GoldenGate architecture 10
Overview of process types 16
Overview of groups 16
Overview of the Commit Sequence Number (CSN) 17
Chapter 2 **Configuring Manager and Network Communications** 18
Overview of the Manager process 18
Assigning Manager a port for local communication 18
Maintaining ports for remote connections through firewalls 18
Choosing an internet protocol 19
Recommended Manager parameters 20
Creating the Manager parameter file 20
Starting Manager 21
Stopping Manager 22
Chapter 3 **Getting started with the Oracle GoldenGate process interfaces** 23
Using the GGSCI command-line interface 23
Using UNIX batch and shell scripts 24
Using Oracle GoldenGate parameter files 25
Supported characters in object names 33
Specifying object names in Oracle GoldenGate input 35
Applying SQL-92 rules for names and literals in parameter files 40
Chapter 4 **Oracle GoldenGate globalization support** 41
Preserving the character set 41
Using Unicode and native characters 42

Chapter 5 Using Oracle GoldenGate for live reporting 43
Overview of the reporting configuration 43
Considerations when choosing a reporting configuration 44
Creating a standard reporting configuration 45
Creating a reporting configuration with a data pump on the source system 47
Creating a reporting configuration with a data pump on an intermediary system. . . . 50
Creating a cascading reporting configuration 55
Chapter 6 Using Oracle GoldenGate for real-time data distribution 63
Overview of the data-distribution configuration. 63
Considerations for a data-distribution configuration. 63
Creating a data distribution configuration 65
Chapter 7 Configuring Oracle GoldenGate for real-time data warehousing 70
Overview of the data-warehousing configuration 70
Considerations for a data warehousing configuration. 70
Creating a data warehousing configuration 72
Chapter 8 Using Oracle GoldenGate to maintain a live standby database 77
Overview of a live standby configuration. 77
Considerations for a live standby configuration 78
Creating a live standby configuration 80
Moving user activity in a planned switchover 86
Moving user activity in an unplanned failover 89
Chapter 9 Using Oracle GoldenGate for active-active high availability 93
Overview of an active-active configuration. 93
Considerations for an active-active configuration. 93
Preventing data looping. 96
Creating an active-active configuration 98
Managing conflicts. 105
Handling conflicts with the Oracle GoldenGate CDR feature. 106
Configuring Oracle GoldenGate CDR. 106
CDR example 1: All conflict types with USEMAX, OVERWRITE, DISCARD 112
CDR example 2: UPDATEROWEXISTS with USEDELTA and USEMAX 119
CRD example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE. 121
Chapter 10 Configuring Oracle GoldenGate security 125
Overview of Oracle GoldenGate security options 125
Encrypting a trail or file. 126
Encrypting a database user password. 130
Encrypting data sent across TCP/IP 133

Generating encryption keys . . . 134
Configuring command security . . . 136
Using target system connection initiation . . . 138

Chapter 11 Mapping and manipulating data . . . 142

Limitations of support . . . 142
Parameters that control mapping and data integration . . . 142
Mapping between dissimilar databases . . . 142
Deciding where data mapping and conversion will take place . . . 143
Globalization considerations when mapping data . . . 143
Mapping columns . . . 146
Selecting rows . . . 153
Retrieving before values . . . 157
Selecting columns . . . 158
Selecting and converting SQL operations . . . 158
Using transaction history . . . 159
Testing and transforming data . . . 160
Using tokens . . . 166

Chapter 12 Handling Oracle GoldenGate processing errors . . . 168

Overview of Oracle GoldenGate error handling . . . 168
Handling Extract errors . . . 168
Handling Replicat errors during DML operations . . . 168
Handling Replicat errors during DDL operations . . . 172
Handling TCP/IP errors . . . 172
Maintaining updated error messages . . . 173
Resolving Oracle GoldenGate errors . . . 173

Chapter 13 Associating replicated data with metadata . . . 174

Configuring Oracle GoldenGate to assume identical metadata . . . 174
Configuring Oracle GoldenGate to assume dissimilar metadata . . . 175
Configuring Oracle GoldenGate to use a combination of similar and dissimilar definitions183

Chapter 14 Configuring online change synchronization . . . 184

Overview of online change synchronization . . . 184
Naming conventions for groups . . . 185
Creating a checkpoint table . . . 185
Creating an online Extract group . . . 187
Creating a trail . . . 190
Creating a parameter file for online extraction . . . 191
Creating an online Replicat group . . . 194

Creating a parameter file for online replication . . . 195
Controlling online processes . . . 196
Deleting a process group . . . 198

Chapter 15 Running an initial data load . . . 200

Overview of initial data load methods . . . 200
Using parallel processing in an initial load . . . 201
Prerequisites for initial load . . . 201
Loading data with a database utility . . . 203
Loading data from file to Replicat . . . 204
Loading data from file to database utility . . . 209
Loading data with an Oracle GoldenGate direct load . . . 214
Loading data with a direct bulk load to SQL*Loader . . . 219
Loading data with Teradata load utilities . . . 224

Chapter 16 Customizing Oracle GoldenGate processing . . . 226

Overview of custom processing . . . 226
Executing commands, stored procedures, and queries with SQLEXEC . . . 226
Using Oracle GoldenGate macros to simplify and automate work . . . 233
Using user exits to extend Oracle GoldenGate capabilities . . . 239
Using the Oracle GoldenGate event marker system to raise database events . . . 242

Chapter 17 Monitoring Oracle GoldenGate processing . . . 247

Overview of the Oracle GoldenGate monitoring tools . . . 247
Using the information commands in GGSCI . . . 247
Monitoring an Extract recovery . . . 248
Monitoring lag . . . 249
Monitoring processing volume . . . 250
Using the error log . . . 252
Using the process report . . . 253
Using the discard file . . . 255
Using the system logs . . . 256
Reconciling time differences . . . 257
Sending event messages to a NonStop system . . . 257
Getting more help with monitoring and tuning . . . 259

Chapter 18 Performing administrative operations . . . 260

Overview of administrative operations . . . 260
Performing application patches . . . 260
Adding process groups . . . 261
Initializing the transaction logs . . . 268

Shutting down the system . . . 269
Changing database attributes . . . 269
Changing the size of trail files . . . 275

Chapter 19 Undoing data changes with the Reverse utility . . . 277

Overview of the Reverse utility . . . 277
Reverse utility restrictions . . . 278
Configuring the Reverse utility . . . 278
Creating process groups and trails for reverse processing . . . 281
Running the Reverse utility . . . 283
Undoing the changes made by the Reverse utility . . . 284

Appendix 1 Supported character sets . . . 285

Appendix 2 Supported locales . . . 292

Appendix 3 About the Oracle GoldenGate trail . . . 295

Trail recovery mode . . . 295
Trail file header record . . . 295
Trail record format . . . 296
Example of an Oracle GoldenGate record . . . 296
Record header area . . . 298
Record data area . . . 300
Tokens area . . . 302
Oracle GoldenGate operation types . . . 302
Oracle GoldenGate trail header record . . . 306

Appendix 4 About the commit sequence number . . . 307

Glossary . . . 310

Index . . . 323

PREFACE

About the Oracle GoldenGate guides

The complete Oracle GoldenGate documentation set contains the following components:

HP NonStop platforms

- *Oracle GoldenGate HP NonStop Administrator's Guide:* Explains how to plan for, configure, and implement the Oracle GoldenGate replication solution on the NonStop platform.
- *Oracle GoldenGate HP NonStop Reference Guide:* Contains detailed information about Oracle GoldenGate parameters, commands, and functions for the NonStop platform.

Windows, UNIX, Linux platforms

- *Installation and Setup guides*: There is one such guide for each database that is supported by Oracle GoldenGate. It contains system requirements, pre-installation and post-installation procedures, installation instructions, and other system-specific information for installing the Oracle GoldenGate replication solution.
- *Oracle GoldenGate Windows and UNIX Administrator's Guide*: Explains how to plan for, configure, and implement the Oracle GoldenGate replication solution on the Windows and UNIX platforms.
- *Oracle GoldenGate Windows and UNIX Reference Guide*: Contains detailed information about Oracle GoldenGate parameters, commands, and functions for the Windows and UNIX platforms.
- *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide:* Contains suggestions for improving the performance of the Oracle GoldenGate replication solution and provides solutions to common problems.

Other Oracle GoldenGate products

- *Oracle GoldenGate Director Administrator's Guide*: Expains how to install, run, and administer Oracle GoldenGate Director for configuring, managing, monitoring, and reporting on the Oracle GoldenGate replication components.
- *Oracle GoldenGate Veridata Administrator's Guide*: Explains how to install, run, and administer the Oracle GoldenGate Veridata data comparison solution.
- *Oracle GoldenGate for Java Administrator's Guide*: Explains how to install, configure, and run Oracle GoldenGate for Java to capture JMS messages to Oracle GoldenGate trails or deliver captured data to messaging systems or custom APIs.
- *Oracle GoldenGate for Flat File Administrator's Guide*: Explains how to install, configure, and run Oracle GoldenGate for Flat File to format data captured by Oracle GoldenGate as batch input to ETL, proprietary or legacy applications.

Typographic conventions used in this manual

This manual uses the following style conventions.

- Parameter and command arguments are shown in upper case, for example:
  ```
  CHECKPARAMS
  ```
- File names, table names, and other names are shown in lower case unless they are case-sensitive to the operating system or software application they are associated with, for example:
  ```
  account_tab
  GLOBALS
  ```
- Variables are shown within < > characters, for example:
  ```
  <group name>
  ```
- When one of multiple mutually-exclusive arguments must be selected, the selection is enclosed within braces and separated with pipe characters, for example:
  ```
  VIEW PARAMS {MGR | <group> | <file name>}
  ```
- Optional arguments are enclosed within brackets, for example:
  ```
  CLEANUP EXTRACT <group name> [, SAVE <count>]
  ```
- When there are numerous multiple optional arguments, a placeholder such as `[<option>]` may be used, and the options are listed and described separately, for example:
  ```
  TRANLOGOPTIONS [<option>]
  ```
- When an argument is accepted more than once, an ellipsis character (...) is used, for example:
  ```
  PARAMS ([<requirement rule>] <param spec> [, <param spec>] [, ...])
  ```
- The ampersand (&) is used as a continuation character in Oracle GoldenGate parameter files. It is required to be placed at the end of each line of a parameter statement that spans multiple lines. Most examples in this documentation show the ampersand in its proper place; however, some examples of multi-line statements may omit it to allow for space constraints of the publication format.

Getting more help with Oracle GoldenGate

In addition to the Oracle GoldenGate documentation, you can get help for Oracle GoldenGate in the following ways.

Getting help with the Oracle GoldenGate interface

Both GGSCI and the Oracle GoldenGate Director applications provide online help.

GGSCI commands

To get help for an Oracle GoldenGate command, use the HELP command in GGSCI. To get a summary of command categories, issue the HELP command without options. To get help

for a specific command, issue the HELP command with the command name as input.

```
HELP <command name>
```

Example:

```
HELP ADD EXTRACT
```

The help file displays the syntax and description of the command.

Oracle GoldenGate Director and Oracle GoldenGate Monitor

To get help for the Oracle GoldenGate graphical client interfaces, use the **Help** menu within the application.

Getting help with questions and problems

For troubleshooting assistance, see *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide*. Additional information can be obtained from the Knowledge Base on http://support.oracle.com. If you cannot find an answer, you can open a service request from the support site.

CHAPTER 1

Introduction to Oracle GoldenGate

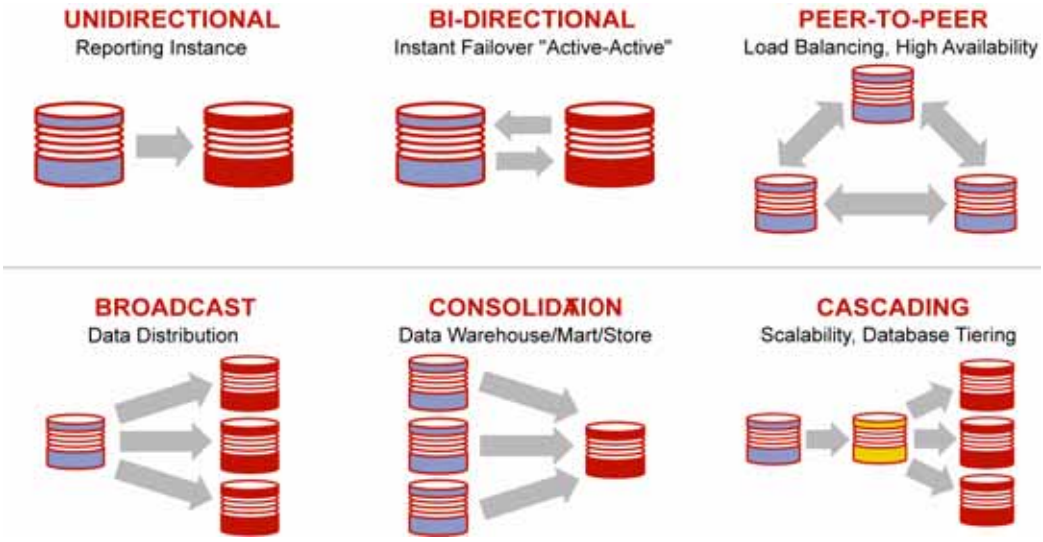
Oracle GoldenGate supported processing methods and databases

Oracle GoldenGate enables the exchange and manipulation of data at the transaction level among multiple, heterogeneous platforms across the enterprise[1]. Its modular architecture gives you the flexibility to extract and replicate selected data records, transactional changes, and changes to DDL (data definition language[2]) across a variety of topologies.

With this flexibility, and the filtering, transformation, and custom processing features of Oracle GoldenGate, you can support numerous business requirements:

- Business continuance and high availability.
- Initial load and database migration.
- Data integration.
- Decision support and data warehousing.

Figure 1 Oracle GoldenGate supported topologies



1. Support for replication across different database types and topologies varies by database type. See the Oracle GoldenGate Installation and Setup Guide for your database for detailed information about supported configurations.

2. DDL is not supported for all databases

Table 1 Supported processing methods[1]

Database	Log-Based Extraction (capture)	Non-Log-Based Extraction** (capture)	Replication (delivery)
DB2 for i*			X
DB2 for Linux, UNIX, Windows	X		X
DB2 for z/OS	X		X
Oracle	X		X
MySQL	X		X
SQL/MX	X		X
SQL Server	X		X
Sybase	X		X
Teradata		X	X
TimesTen*			X
Generic ODBC*			X

[1] For full information about processing methodology, supported topologies and functionality, and configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database.

*Supported only as a target database. Cannot be a source database for Oracle GoldenGate extraction.

** Uses a capture module that communicates with the Oracle GoldenGate API to send change data to Oracle GoldenGate.

*** Only like-to-like configuration is supported. Data manipulation, filtering, column mapping not supported.

Overview of the Oracle GoldenGate architecture

Oracle GoldenGate can be configured for the following purposes:

- A static extraction of data records from one database and the loading of those records to another database.
- Continuous extraction and replication of transactional DML[1] operations and DDL changes (for supported databases) to keep source and target data consistent.
- Extraction from a database and replication to a file outside the database.

Oracle GoldenGate is composed of the following components:

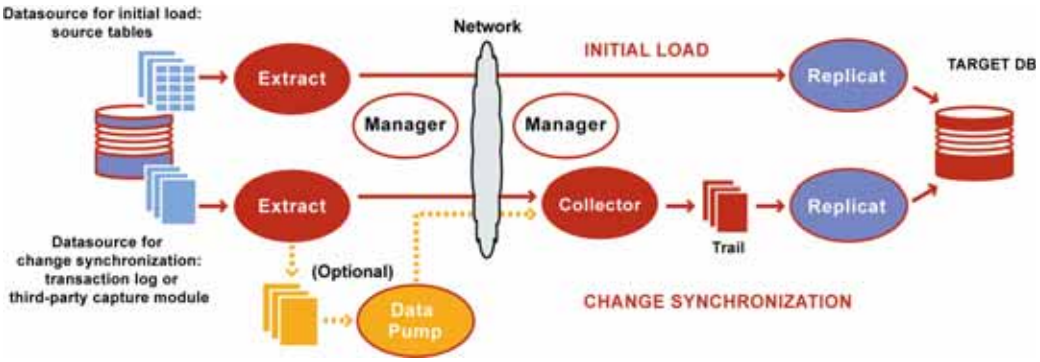
- Extract

1. INSERT, UPDATE, DELETE

- Data pump
- Replicat
- Trails or extract files
- Checkpoints
- Manager
- Collector

Figure 2 illustrates the logical architecture of Oracle GoldenGate for initial data loads and for the synchronization of DML and DDL operations. This is the basic configuration. Variations of this model are recommended depending on business needs.

Figure 2 Oracle GoldenGate logical architecture



Overview of Extract

The Extract process runs on the source system and is the extraction (capture) mechanism of Oracle GoldenGate. You can configure Extract in one of the following ways:

- **Initial loads:** For initial data loads, Extract extracts (captures) a current, static set of data directly from their source objects.
- **Change synchronization:** To keep source data synchronized with another set of data, Extract captures DML and DDL operations after the initial synchronization has taken place.

Extract captures from a *data source* that can be one of the following:

- Source tables, if the run is an initial load.
- The *database recovery logs or transaction logs* (such as the Oracle redo logs or SQL/MX audit trails). The actual method of capturing from the logs varies depending on the database type.
- A *third-party capture module*. This method provides a communication layer that passes data and metadata from an external API to the Extract API. The database vendor or a third-party vendor provides the components that extract the data operations and pass them to Extract.

When configured for change synchronization, Extract captures the DML and DDL operations that are performed on objects in the Extract configuration. Extract stores these

operations until it receives commit records or rollbacks for the transactions that contain them. When a rollback is received, Extract discards the operations for that transaction. When a commit is received, Extract persists the transaction to disk in a series of files called a *trail*, where it is queued for propagation to the target system. All of the operations in each transaction are written to the trail as a sequentially organized transaction unit. This design ensures both speed and data integrity.

NOTE Extract ignores operations on objects that are not in the Extract configuration, even though the same transaction may also include operations on objects that are in the Extract configuration.

Multiple Extract processes can operate on different objects at the same time. For example, two Extract processes can extract and transmit in parallel to two Replicat processes (with two persistence trails) to minimize target latency when the databases are large. To differentiate among different Extract processes, you assign each one a group name (see "Overview of groups" on page 16).

Overview of data pumps

A data pump is a secondary Extract group within the source Oracle GoldenGate configuration. If a data pump is not used, Extract must send the captured data operations to a remote trail on the target. In a typical configuration with a data pump, however, the primary Extract group writes to a trail on the source system. The data pump reads this trail and sends the data operations over the network to a remote trail on the target. The data pump adds storage flexibility and also serves to isolate the primary Extract process from TCP/IP activity.

In general, a data pump can perform data filtering, mapping, and conversion, or it can be configured in *pass-through mode*, where data is passively transferred as-is, without manipulation. Pass-through mode increases the throughput of the data pump, because all of the functionality that looks up object definitions is bypassed.

In most business cases, you should use a data pump. Some reasons for using a data pump include the following:

- **Protection against network and target failures:** In a basic Oracle GoldenGate configuration, with only a trail on the target system, there is nowhere on the source system to store the data operations that Extract continuously extracts into memory. If the network or the target system becomes unavailable, Extract could run out of memory and abend. However, with a trail and data pump on the source system, captured data can be moved to disk, preventing the abend of the primary Extract. When connectivity is restored, the data pump captures the data from the source trail and sends it to the target system(s).
- **You are implementing several phases of data filtering or transformation.** When using complex filtering or data transformation configurations, you can configure a data pump to perform the first transformation either on the source system or on the target system, or even on an intermediary system, and then use another data pump or the Replicat group to perform the second transformation.
- **Consolidating data from many sources to a central target.** When synchronizing multiple source databases with a central target database, you can store extracted data operations on each source system and use data pumps on each of those systems to send

the data to a trail on the target system. Dividing the storage load between the source and target systems reduces the need for massive amounts of space on the target system to accommodate data arriving from multiple sources.

- **Synchronizing one source with multiple targets.** When sending data to multiple target systems, you can configure data pumps on the source system for each target. If network connectivity to any of the targets fails, data can still be sent to the other targets.

Overview of Replicat

The Replicat process runs on the target system, reads the trail on that system, and then reconstructs the DML or DDL operations and applies them to the target database. You can configure Replicat in one of the following ways:

- **Initial loads:** For initial data loads, Replicat can apply a static data copy to target objects or route it to a high-speed bulk-load utility.
- **Change synchronization:** When configured for change synchronization, Replicat applies the replicated source operations to the target objects using a native database interface or ODBC, depending on the database type. To preserve data integrity, Replicat applies the replicated operations in the same order as they were committed to the source database.

You can use multiple Replicat processes with multiple Extract processes in parallel to increase throughput. To preserve data integrity, each set of processes handles a different set of objects. To differentiate among Replicat processes, you assign each one a group name (see "Overview of groups" on page 16).

You can delay Replicat so that it waits a specific amount of time before applying the replicated operations to the target database. A delay may be desirable, for example, to prevent the propagation of errant SQL, to control data arrival across different time zones, or to allow time for other planned events to occur. The length of the delay is controlled by the DEFERAPPLYINTERVAL parameter.

Overview of trails

To support the continuous extraction and replication of database changes, Oracle GoldenGate stores records of the captured changes temporarily on disk in a series of files called a *trail*. A trail can exist on the source system, an intermediary system, the target system, or any combination of those systems, depending on how you configure Oracle GoldenGate. On the local system it is known as an *extract trail* (or *local trail*). On a remote system it is known as a *remote trail*.

By using a trail for storage, Oracle GoldenGate supports data accuracy and fault tolerance (see "Overview of checkpoints" on page 14). The use of a trail also allows extraction and replication activities to occur independently of each other. With these processes separated, you have more choices for how data is processed and delivered. For example, instead of extracting and replicating changes continuously, you could extract changes continuously but store them in the trail for replication to the target later, whenever the target application needs them.

Processes that write to, and read, a trail

The primary Extract and the data-pump Extract write to a trail. Only one Extract process can write to a trail, and each Extract must be linked to a trail.

Processes that read the trail are:

- Data-pump Extract: Extracts DML and DDL operations from a local trail that is linked to a previous Extract (typically the primary Extract), performs further processing if needed, and transfers the data to a trail that is read by the next Oracle GoldenGate process downstream (typically Replicat, but could be another data pump if required).
- Replicat: Reads the trail and applies replicated DML and DDL operations to the target database.

Trail creation and maintenance

The trail files themselves are created as needed during processing, but you specify a two-character name for the trail when you add it to the Oracle GoldenGate configuration with the ADD RMTTRAIL or ADD EXTTRAIL command. By default, trails are stored in the dirdat sub-directory of the Oracle GoldenGate directory.

Full trail files are aged automatically to allow processing to continue without interruption for file maintenance. As each new file is created, it inherits the two-character trail name appended with a unique, six-digit sequence number from 000000 through 999999 (for example c:\ggs\dirdat\tr000001). When the sequence number reaches 999999, the numbering starts over at 000000.

You can create more than one trail to separate the data from different objects or applications. You link the objects that are specified in a TABLE or SEQUENCE parameter to a trail that is specified with an EXTTRAIL or RMTTRAIL parameter in the Extract parameter file. Aged trail files can be purged by using the Manager parameter PURGEOLDEXTRACTS.

To maximize throughput, and to minimize I/O load on the system, extracted data is sent into and out of a trail in large blocks. Transactional order is preserved. By default, Oracle GoldenGate writes data to the trail in *canonical format*, a proprietary format which allows it to be exchanged rapidly and accurately among heterogeneous databases. However, data can be written in other formats that are compatible with different applications.

For additional information about the trail and the records it contains, see Appendix 2 on page 562.

Overview of extract files

In some configurations, Oracle GoldenGate stores extracted data in an *extract file* instead of a trail. The extract file can be a single file, or it can be configured to roll over into multiple files in anticipation of limitations on file size that are imposed by the operating system. In this sense, it is similar to a trail, except that checkpoints are not recorded. The file or files are created automatically during the run. The same versioning features that apply to trails also apply to extract files.

Overview of checkpoints

Checkpoints store the current read and write positions of a process to disk for recovery purposes. Checkpoints ensure that data changes that are marked for synchronization actually are captured by Extract and applied to the target by Replicat, and they prevent redundant processing. They provide fault tolerance by preventing the loss of data should the system, the network, or an Oracle GoldenGate process need to be restarted. For complex synchronization configurations, checkpoints enable multiple Extract or Replicat processes to read from the same set of trails.

Checkpoints work with inter-process acknowledgments to prevent messages from being lost in the network. Oracle GoldenGate has a proprietary guaranteed-message delivery technology.

Extract creates checkpoints for its positions in the data source and in the trail. Because Extract only captures committed transactions, it must keep track of operations in all open transactions, in the event that any of them are committed. This requires Extract to record a checkpoint where it is currently reading in a transaction log, plus the position of the start of the oldest open transaction, which can be in the current or any preceding log.

To control the amount of transaction log that must be re-processed after an outage, Extract persists the current state and data of processing to disk at specific intervals, including the state and data (if any) of long-running transactions. If Extract stops after one of these intervals, it can recover from a position within the previous interval or at the last checkpoint, instead of having to return to the log position where the oldest open long-running transaction first appeared. For more information, see the BR parameter in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Replicat creates checkpoints for its position in the trail. Replicat stores its checkpoints in a checkpoint table in the target database to couple the commit of its transaction with its position in the trail file. The checkpoint table guarantees consistency after a database recovery by ensuring that a transaction will only be applied once, even if there is a failure of the Replicat process or the database process. For reporting purposes, Replicat also has a checkpoint file on disk in the dirchk sub-directory of the Oracle GoldenGate directory.

Checkpoints are not required for non-continuous types of configurations that can be re-run from a start point if needed, such as initial loads.

Overview of Manager

Manager is the control process of Oracle GoldenGate. Manager must be running on each system in the Oracle GoldenGate configuration before Extract or Replicat can be started, and Manager must remain running while those processes are running so that resource management functions are performed. Manager performs the following functions:

- Start Oracle GoldenGate processes
- Start dynamic processes
- Maintain port numbers for processes
- Perform trail management
- Create event, error, and threshold reports

One Manager process can control many Extract or Replicat processes. On Windows systems, Manager can run as a service. For more information about the Manager process and configuring TCP/IP connections, see Chapter 2.

Overview of Collector

Collector is a process that runs in the background on the target system when continuous, online change synchronization is active. Collector does the following:

- Upon a connection request from a remote Extract to Manger, scan and bind to an available port and then send the port number to Manager for assignment to the requesting Extract process.

- Receive extracted database changes that are sent by Extract and write them to a trail file. Manager starts Collector automatically when a network connection is required, so Oracle GoldenGate users do not interact with it. Collector can receive information from only one Extract process, so there is one Collector for each Extract that you use. Collector terminates when the associated Extract process terminates.

  NOTE Collector can be run manually, if needed. This is known as a *static* Collector (as opposed to the regular, *dynamic* Collector). Several Extract processes can share one static Collector; however, a one-to-one ratio is optimal. A static Collector can be used to ensure that the process runs on a specific port. For more information about the static Collector, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. For more information about how Manager assigns ports, see Chapter 2.

By default, Extract initiates TCP/IP connections from the source system to Collector on the target, but Oracle GoldenGate can be configured so that Collector initiates connections from the target. Initiating connections from the target might be required if, for example, the target is in a trusted network zone, but the source is in a less trusted zone. For information about this configuration, see page 138.

Overview of process types

Depending on the requirement, Oracle GoldenGate can be configured with the following processing types.

- An *online* Extract or Replicat process runs until stopped by a user. Online processes maintain recovery checkpoints in the trail so that processing can resume after interruptions. You use online processes to continuously extract and replicate DML and DDL operations (where supported) to keep source and target objects synchronized. The EXTRACT and REPLICAT parameters apply to this process type.
- A *source-is-table* Extract process extracts a current set of static data directly from the source objects in preparation for an initial load to another database. This process type does not use checkpoints. The SOURCEISTABLE parameter applies to this process type.
- A *special-run* Replicat process applies data within known begin and end points. You use a special Replicat run for initial data loads, and it also can be used with an online Extract to apply data changes from the trail in batches, such as once a day rather than continuously. This process type does not maintain checkpoints, because the run can be started over with the same begin and end points. The SPECIALRUN parameter applies to this process type.
- A *remote task* is a special type of initial-load process in which Extract communicates directly with Replicat over TCP/IP. Neither a Collector process nor temporary disk storage in a trail or file is used. The task is defined in the Extract parameter file with the RMTTASK parameter.

Overview of groups

To differentiate among multiple Extract or Replicat processes on a system, you define processing *groups*. For example, to replicate different sets of data in parallel, you would create two Replicat groups.

A processing group consists of a process (either Extract or Replicat), its parameter file, its

checkpoint file, and any other files associated with the process. For Replicat, a group also includes the associated checkpoint table.

You define groups by using the ADD EXTRACT and ADD REPLICAT commands in the Oracle GoldenGate command interface, GGSCI. For permissible group names, see those commands in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

All files and checkpoints relating to a group share the name that is assigned to the group itself. Any time that you issue a command to control or view processing, you supply a group name or multiple group names by means of a wildcard.

Overview of the Commit Sequence Number (CSN)

When working with Oracle GoldenGate, you might need to refer to a *Commit Sequence Number*, or CSN. A CSN is an identifier that Oracle GoldenGate constructs to identify a transaction for the purpose of maintaining transactional consistency and data integrity. It uniquely identifies a point in time in which a transaction commits to the database.

The CSN can be required to position Extract in the transaction log, to reposition Replicat in the trail, or for other purposes. It is returned by some conversion functions and is included in reports and certain GGSCI output.

For more information about the CSN and a list of CSN values per database, see Appendix 1 on page 559.

CHAPTER 2

Configuring Manager and Network Communications

This chapter contains instructions for:

- Configuring the Manager process.
- Specifying ports for local and remote network communications. All Oracle GoldenGate ports are configurable.

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. The Manager process performs the following functions:

- Start Oracle GoldenGate processes
- Start dynamic processes
- Start the Collector process
- Manage the port numbers for processes
- 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.

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.
- 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" on page 172.

For more information about PORT and DYNAMICPORTLIST, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Recommended Manager parameters

The following parameters are optional, but recommended, for the Manager process. For more information about these and additional Manager parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

- 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.
- 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.
- 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.
- 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.
- USERID: Required if using Oracle GoldenGate DDL support.

Creating the Manager parameter file

To configure Manager with required port information and optional parameters, create a parameter file by following these steps. For more information about Oracle GoldenGate parameter files, see Chapter 3 on page 23.

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 Installation and Setup guide 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. If a parameter statement must span multiple lines, use an ampersand (&) before each line break.

4. Save, then close the file.

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

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

For detailed information about these and other Manger parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Starting Manager

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

- The command line of any supported operating system.
- The GGSCI command interface.
- 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.

To start Manager from the command shell of the operating system

```
mgr paramfile <param 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.

To start Manager from GGSCI

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

```
START MANAGER
```

NOTE When starting Manager from the command line or GGSCI on Windows Server 2008 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.

To stop Manager

- 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.
- 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.
- 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.

CHAPTER 3

Getting started with the Oracle GoldenGate process interfaces

Oracle GoldenGate users provide instructions to the processes in the following ways:

- GGSCI (Oracle GoldenGate Software Command Interface)
- Batch and shell scripts
- Parameter files

Using the GGSCI command-line interface

GGSCI is the Oracle GoldenGate command-line interface. You can use GGSCI to issue the complete range of commands that configure, control, and monitor Oracle GoldenGate.

To start GGSCI

1. Change directories to the one where Oracle GoldenGate is installed.
2. Run the ggsci executable file.

For more information about Oracle GoldenGate commands, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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, given a table that bears the Euro symbol as its name, the following commands are equivalent:

```
ADD TRANDATA €1
ADD TRANDATA \u20AC1
```

For more information, see "Using escape sequences for specific characters" on page 144.

NOTE Oracle GoldenGate group names are case-insensitive.

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*
```

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 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. For more information, see "Using escape sequences for specific characters" on page 144.

To use OBEY

1. Create and save a text file that contains the commands, one command per line. This is your OBEY file. Name it what you want. 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.

   ```
   ALLOWNESTED
   ```

4. In GGSCI, call the OBEY file by using the following syntax.

   ```
   OBEY <file name>
   ```

   Where: <file name> is the relative or fully qualified name of the OBEY file.

Figure 3 illustrates an OBEY command file for use with the OBEY command. It creates and starts Extract and Replicat groups and retrieves processing information.

Figure 3 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
```

Using UNIX batch and shell scripts

On a UNIX system, 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" on page 23.

To input a script

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

```
ggsci < <input_file>
```

Where: <input_file> is a text file containing the commands that you want to issue, in the order they are to be issued, and the "<" character pipes the file into the GGSCI program.

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 processing to enter into a loop.

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.

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 specify characters that are not compatible with the character set of the operating system where the file will be used. See "Using escape sequences for specific characters" on page 144.

For more information about this parameter, see the Oracle GoldenGate *Windows and UNIX Reference Guide*

Overview of 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.

Overview of 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” on page 31.

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. USERID in Figure 4 and Figure 5 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 Figure 4 and Figure 5. 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 Figure 5 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.

The following are examples of basic parameter files for Extract and Replicat. Comments are preceded by double hyphens.

Figure 4 Sample Extract parameter file

```
-- Extract group name
EXTRACT capt
-- Extract database user login, with password encryption specifications
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
-- Discard file
DISCARDFILE /ggs/capt.dsc, PURGE
-- Remote host to where captured data is sent in encrypted format:
RMTHOST sysb, MGRPORT 7809, ENCRYPT AES192 KEYNAME mykey
-- Encryption specification for trail data
ENCRYPTTRAIL AES192 KEYNAME mykey1
-- Remote trail on the remote host
RMTTRAIL /ggs/dirdat/aa
-- TABLE statements that identify data to capture.
TABLE FIN.*;
TABLE SALES.*;
```

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” on page 35 for more information.

Figure 5 Sample Replicat parameter file

```
-- Replicat group name
REPLICAT deliv
-- Replicat database user login, with password encryption specifications
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
-- File containing definitions of source objects
SOURCEDEFS /ggs/dirdef/defs
-- Discard file
DISCARDFILE /ggs/deliv.dsc, PURGE
-- Decryption specification for encrypted trail
DECRYPTTRAIL AES192 KEYNAME mykey1
-- 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');
```

Note the use of single and double quote marks in the Replicat example in Figure 5. 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. Note that to specify case-sensitive column names in double quotes, you must use the USEANSISQLQUOTES parameter in the GLOBALS file; otherwise strings in double quotes are interpreted as literals. For more information about specifying names and literals, see "Specifying object names in Oracle GoldenGate input" on page 35.

Creating a parameter file

To create a parameter file, use the EDIT PARAMS command within the GGSCI user interface (recommended) 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.

Use GGSCI only if you are writing the parameter file in the character set of the operating system (no CHARSET parameter is being used). Otherwise, use a text editor outside GGSCI. For more information about CHARSET, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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

  NOTE You can change the default editor through the GGSCI interface by using the SET EDITOR command.

To create a parameter file in GGSCI

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 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> <argument> [, <option>] [&]
   ```

   Where:

   - <PARAMETER> is the name of the parameter.
   - <argument> is a required argument for the parameter. Some parameters take arguments, but others do not. Separate all arguments with commas, as in the following example:

     ```
     USERID ogg, PASSWORD &
        AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
        AES128 KEYNAME mykey1
     RMTHOST sysb, MGRPORT 8040, ENCRYPT AES192 KEYNAME mykey
     ENCRYPTTRAIL AES 192 KEYNAME mykey2
     RMTTRAIL /home/ggs/dirdat/c1, PURGE
     ```

   - [, <option>] is an optional argument.
   - [&] is required at the end of each line in a multi-line parameter statement, as in the USERID 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
```

5. Save and close the file.

To create 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. See also "Storing parameter files".

Storing parameter files

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 by specifying the full path name, but you also must specify the full path name with the PARAMS option of the ADD EXTRACT or ADD REPLICAT command when you create the process group. 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.

Verifying a parameter file

You can check the syntax of parameters in an Extract or Replicat parameter file for accuracy. This feature is not available for other Oracle GoldenGate processes.

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>
   ```

   Oracle GoldenGate audits the syntax and writes the results to the report file or to the screen. Then the process stops. For more information about the report file, see Chapter 17.

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.

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.

WARNING 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 editing the parameter file, and then started again after saving the parameter file. Changing parameter settings while a process is running can have unpredictable and adverse consequences, especially if you are adding tables or changing mapping or filtering rules.

WARNING 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 using the following command in GGSCI, unless you want to stop Manager in a Windows cluster; in that case, Manager must be stopped by using 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 (use the Cluster Administrator if starting Manager in a Windows cluster).

   ```
   START {EXTRACT | REPLICAT | MANAGER} <group name>
   ```

Simplifying the creation of parameter files

Oracle GoldenGate provides tools that reduce the number of times that a parameter must be specified.

- Wildcards
- The OBEY parameter
- Macros

- Parameter substitution

Using wildcards

For parameters that accept object names, you can use an asterisk (*) wildcard to match any number of characters. Owner names, if used, cannot be specified with wildcards. The use of wildcards reduces the work of specifying numerous object names or all objects within a given schema.

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 honored to read wildcarded object names in the referenced file, but you must use an escape sequence (\uXXXX) for all other multibyte specifications in the referenced file. For more information about CHARSET, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Using macros

You can use macros to automate multiple uses of a parameter statement. For more information, see “Using Oracle GoldenGate macros to simplify and automate work” on page 233.

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, preceding the runtime parameter name with a question mark (?) as shown in the following example.

```
SOURCEISFILE
EXTFILE ?EXTFILE
MAP ?TABNAME, TARGET 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 Figure 6 and Figure 7.

Figure 6 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
```

Figure 7 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

For more information about Oracle GoldenGate parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Supported characters in object names

Oracle GoldenGate supports most characters in object and column names. 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.

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
*	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 \?)
/	Forward slash
\	Backward slash (Must be \\ when used in parameter file)

Character	Description
@	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

Non-supported special characters

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

Character	Description
{ }	Begin and end curly brackets (braces)
[ ]	Begin and end brackets
=	Equal symbol
+	Plus sign
!	Exclamation point
~	Tilde
|	Pipe
&	Ampersand
:	Colon
;	Semi-colon
,	Comma
‘ ’	Single quotes

Character	Description
“ ”	Double quotes
'	Accent mark (Diacritical mark)
.	Period
<	Less-than symbol
>	Greater-than symbol

Specifying object names in Oracle GoldenGate input

Follow these rules when specifying database object names in parameter files, column-conversion functions, commands, and in other input.

- Object names can be any length and in any supported character set. For supported characters, see “Supported characters in object names” on page 33. For supported character sets, see “Supported character sets” on page 285.
- If an object name is not fully qualified in the parameter file, such as listing only EMP as the table name, Oracle GoldenGate uses the default schema name of the login session. For example, if FIN is the default schema of the current login session, FIN.EMP is used.
- Specify object names in double quote marks if they contain special characters such as white spaces or symbols.
- 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.

The following table 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.

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

Table 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
Oracle	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/MX	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in upper case	Case-sensitive, stores in mixed case
PostgreSQL	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in lower case	Case-sensitive, stores in mixed case
MySQL (Case-sensitive database)	No ◆ Always case-sensitive, stores in mixed case ◆ The names of columns, triggers, and procedures are case-insensitive	No effect	No effect
MySQL (Case-insensitive database)	No ◆ Always case-insensitive, stores in mixed case ◆ The names of columns, triggers, and procedures are case-insensitive	No effect	No effect
◆ **SQL Server** ◆ **Sybase** (Database created as case-sensitive)	No Always case-sensitive, stores in mixed case	No effect	No effect

Table 2 Case sensitivity of object names per database (continued)

Database	Requires quotes to enforce case-sensitivity?	Unquoted object name	Quoted object name
◆ **SQL Server** ◆ **Sybase** (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
TimesTen	No Always case-insensitive, stores in upper case	Case-insensitive, stores in upper case	Case-insensitive, stores in upper case

Using wildcards in object names

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 TAB*n*, where *n* is from 0 to 9, a wildcard specification of HQ.TAB? would return HQ.TAB0, HQ.TAB1, HQ.TAB2, and so on, up to HQ.TAB9, but no others. This wildcard is not supported for DEFGEN.
- 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.
- In TABLE and MAP statements, you can combine the asterisk and question-mark wildcard characters in source object names.
- You can use wildcards for table and sequence names. Do not use a wildcard for an owner name (such as schema).

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

The TABLE, MAP and SEQUENCE parameters takes 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 matching 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 *;
TABLE "*";
```

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 ?;
TABLE "?";
```

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 "abc*";
  TABLE "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 abc*;
  TABLE 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 (unless DDL replication is supported and in use. This 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 produces the following 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_TEST<xxx>. The following illustrates this:

- HQ.T_TEST1 maps to RPT.T_T_TEST1

- HQ.T_TEST2 maps to RPT.T_T_TEST2

(And so forth)

The following examples show the correct use of asterisk wildcarding.

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

This enables the following correct result:

- HQ.T_TEST1 maps to RPT.T_TEST1
- HQ.T_TEST2 maps to RPT.T_TEST2

(And so forth)

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. For more information, see the NAMEMATCHIGNORECASE parameter in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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 and for Sybase. Note that in this example, if the target database is Oracle, DB2 or SQL/MX, 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 "abc", TARGET *;
```

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 abc, TARGET *;
```

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 TABLEEXCLUDE and MAPEXCLUDE parameters.

Applying SQL-92 rules for names and literals in parameter files

In Oracle GoldenGate parameter files, conversion functions, user exits, and commands, characters enclosed within double quotes are treated as string literals by default. By default, case-sensitivity of quoted column names is not recognized (for databases that require quotes to enforce case-sensitivity, such as Oracle). For example, the following are correct, where PRODUCT_CODE is the column name, and the other strings are literals:

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

To support case-sensitive object names in double quotes, use the USEANSISQLQUOTES parameter in the GLOBALS parameter file. This parameter applies SQL-92 rules. When USEANSISQLQUOTES is used, you can specify column names in double quotes, and you must specify literal strings in 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')
```

For more information on USEANSISQLQUOTES, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

NOTE The TABLE, SEQUENCE, and MAP statements support quoted table and sequence names by default without requiring USEANSISQLQUOTES.

CHAPTER 4
Oracle GoldenGate globalization support

Oracle GoldenGate globalization support enables the processing of data in its native language encoding. The following points highlight this support.

Preserving the character set

In order to process the data in its native language encoding, Oracle GoldenGate takes into consideration the character set encoding of the database, and the operating system locale for parsing parameter files and command line interpreter processing.

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.

For additional information about character sets, conversion between them, and data mapping, see “Mapping and manipulating data” on page 142.

Character set of database connection

The Extract and Replicat processes uses a session character set when connecting to the database. For Oracle, the session character set is set to the same as the database character set for the Extract process and is determined using NLS_LANG for the Replicat process. For Sybase, Teradata and MySQL, the session character set is taken from the SESSIONCHARSET option of SOURCEDB and TARGETDB. 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

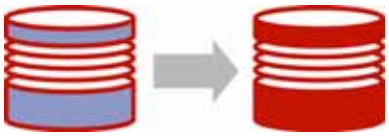
Oracle GoldenGate supports the use of an escape sequence to represent characters in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. You can use an escape sequence if the operating system does not support the required character, or for any other purpose when needed. For more information, see “Using escape sequences for specific characters” on page 144.

CHAPTER 5

Using Oracle GoldenGate for live reporting

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).



Reporting topologies

Oracle GoldenGate supports a number of topologies for reporting that enable you to custom-configure the modules based on your requirements for scalability, availability, and performance.

Single target

- "Creating a standard reporting configuration" on page 45
- "Creating a reporting configuration with a data pump on the source system" on page 47
- "Creating a reporting configuration with a data pump on an intermediary system" on page 50
- "Creating a cascading reporting configuration" on page 55

Multiple targets

You can send data to multiple reporting targets. See "Using Oracle GoldenGate for real-time data distribution" on page 63.

Considerations when choosing a reporting configuration

Data volume

The standard configuration is sufficient if:

- The transaction load is consistent and of moderate volume that is spread out more or less evenly among all of the objects to be replicated.

  and...

- There are none of the following: tables that are subject to long-running transactions, tables that have a very large number of columns that change, or tables that contain columns for which Oracle GoldenGate must fetch from the database (generally columns with LOBs, columns that are affected by SQL procedures executed by Oracle GoldenGate, and columns that are not logged to the transaction log).

If your environment does not satisfy those conditions, consider adding one or more sets of parallel processes. For more information, see the *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide*.

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:

- Native 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" on page 142
- "Customizing Oracle GoldenGate processing" on page 226

Read-only vs. high availability

The Oracle GoldenGate live reporting configuration supports a read-only target. If the target in this configuration will also be used for transactional activity in support of high availability, see "Using Oracle GoldenGate for active-active high availability" on page 93.

Additional information

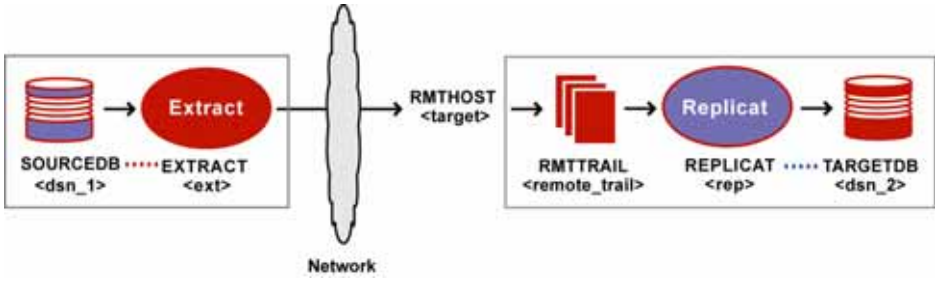
- For additional system and database configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database type.
- For information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see "Configuring online change synchronization" on page 184.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- For more information about tuning this configuration, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

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 8 for a visual representation of the objects you will be creating.

Figure 8 Configuration elements for replication to one target



Source system

To configure the Manager process

1. On the source, configure the Manager process according to the instructions in Chapter 2.

To configure the Extract group

2. 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>
[, THREADS <n>]
```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

3. 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.

4. 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.

```
-- Identify the Extract group:
EXTRACT <ext>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>[, PASSWORD <pw>
   [<encryption options>]]
-- Specify the name or IP address of the target system and
-- optional encryption across TCP/IP:
RMTHOST <target>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the target system:
ENCRYPTTRAIL [<encryption options>]
RMTTRAIL <remote_trail>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

Target system

To configure the Manager process

5. On the target, configure the Manager process according to the instructions in Chapter 2.
6. In the Manager parameter file, use the PURGEOLDEXTRACTS parameter to control the purging of files from the local trail.

To configure the Replicat group

7. On the target, create a Replicat checkpoint table. For instructions, see "Creating a checkpoint table" on page 185. All Replicat groups can use the same checkpoint table.
8. On the target, use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep*.

```
ADD REPLICAT <rep>, EXTTRAIL <remote_trail>, BEGIN <time>
```

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

9. 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.

```
-- Identify the Replicat group:
REPLICAT <rep>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify error handling rules:
-- Specify decryption if trail is encrypted:
DECRYPTTRAIL <encryption options>
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

Creating a reporting configuration with a data pump on the source system

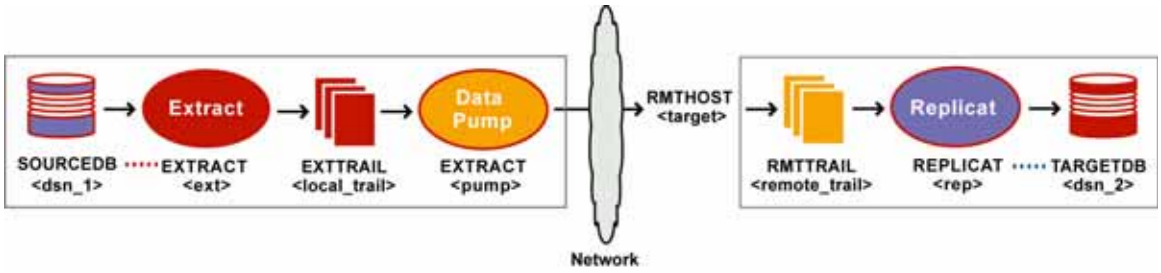
You can add a data pump on the source system to isolate the primary Extract from TCP/IP functions, to add storage flexibility, and to offload the overhead of filtering and conversion processing from the primary Extract.

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 9 for a visual representation of the objects you will be creating.

Figure 9 Configuration elements for replicating to one target with a data pump



Source system

To configure the Manager process

1. On the source, configure the Manager process according to the instructions in Chapter 2.
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

3. On the source, use the ADD EXTRACT command to create a primary Extract group. For documentation purposes, this group is called *ext*.

```
ADD EXTRACT <ext>, {TRANLOG | INTEGRATED TRANLOG}, BEGIN <time>
[, THREADS <n>]
```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

4. 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.

5. 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.

```
-- Identify the Extract group:
EXTRACT <ext>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][,USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to and
-- optional encryption:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

To configure the data pump Extract group

6. 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.

7. 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.

8. 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 if using NOPASSTHRU:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOST <target>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the target system:
ENCRYPTTRAIL [<encryption options>]
RMTTRAIL <remote_trail>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Target system

To configure the Manager process

9. On the target, configure the Manager process according to the instructions in Chapter 2.

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

To configure the Replicat group

11. On the target, create a Replicat checkpoint table. For instructions, see “Creating a checkpoint table” on page 185.

12. On the target, use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep*.

```
ADD REPLICAT <rep>, EXTTRAIL <remote_trail>, BEGIN <time>
```

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

13. 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.

```
-- Identify the Replicat group:
REPLICAT <rep>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

Creating a reporting configuration with a data pump on an intermediary system

You can add a data pump on an intermediary system to add storage flexibility and to offload the overhead of filtering and conversion processing from the source system. You also can use this configuration 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.

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

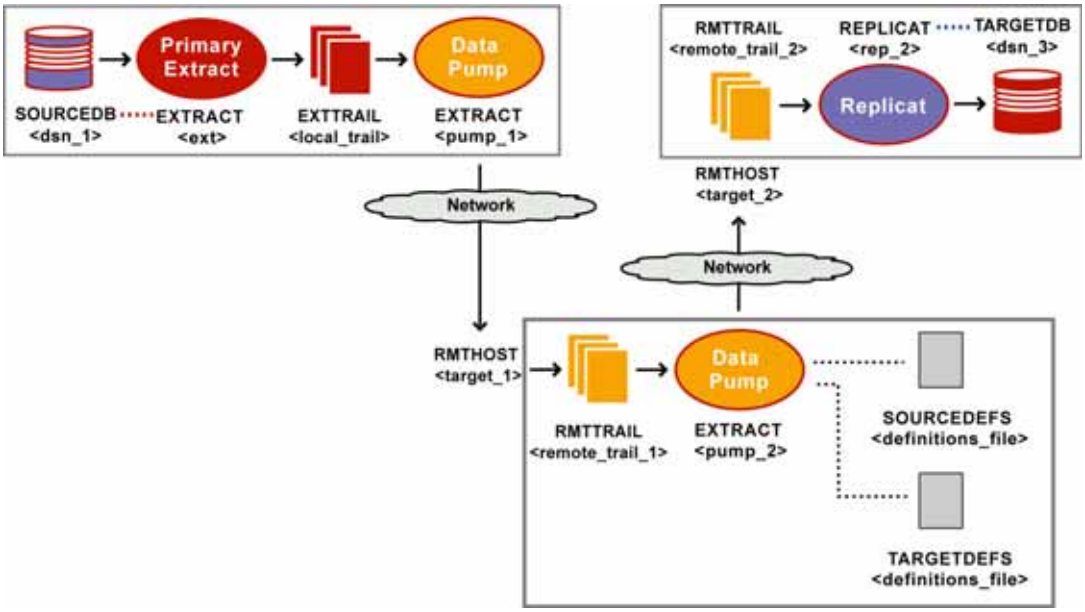
NOTE The data pump on the source system is optional, but will help to protect against data loss in the event of a network outage.

This is a form of cascaded replication. However, in this configuration, data is not applied to a database on the intermediary system. To include a database on the intermediary system in the Oracle GoldenGate configuration, see “Creating a cascading reporting configuration” on page 58.

If you want to use the data pump on the intermediary system to perform conversion and transformation, 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. For more information about this topic, see Chapter 13.

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

Figure 10 Configuration elements for replication through an intermediary system



Source system

To configure the Manager process

1. On the source, configure the Manager process according to the instructions in Chapter 2.
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

3. On the source, use the ADD EXTRACT command to create a primary Extract group. For documentation purposes, this group is called *ext*.

```
ADD EXTRACT <ext>, {TRANLOG | INTEGRATED TRANLOG}, BEGIN <time>
[, THREADS <n>]
```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

4. 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.

5. 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.

```
-- Identify the Extract group:
EXTRACT <ext>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to and
-- encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

To configure the data pump on the source

6. 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.

7. 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.

8. 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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the intermediary system
-- and optional encryption of data over TCP/IP:
RMTHOST <target_1>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify remote trail and encryption options on intermediary system:
ENCRYPTTRAIL [<encryption options>]
RMTTRAIL <remote_trail_1>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Intermediary system

To configure the Manager process on the intermediary system

9. On the intermediary system, configure the Manager process according to the instructions in Chapter 2.
10. 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

11. 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.

12. 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.

13. 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.
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify the target definitions file if SOURCEDEFS was used:
TARGETDEFS <full_pathname>
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOST <target_2>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the target system:
ENCRYPTTRAIL [<encryption options>]
RMTTRAIL <remote_trail_2>
-- Allow mapping, filtering, conversion or pass data through as-is;
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

   - Use SOURCEDEFS and TARGETDEFS to specify the definitions files if the data pump will perform conversion and transformation.
   - Use NOPASSTHRU (the default) if the data pump will perform conversion and transformation. Otherwise, use PASSTHRU.

Target system

To configure the Manager process on the target

14. On the target system, configure the Manager process according to the instructions in Chapter 2.
15. 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

16. On the target, create a Replicat checkpoint table. For instructions, see “Creating a checkpoint table” on page 185.
17. On the target, use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep*.

```
ADD REPLICAT <rep>, EXTTRAIL <remote_trail_2>, BEGIN <time>
```

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

18. 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.

```
-- Identify the Replicat group:
REPLICAT <rep>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

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. Use this configuration if:

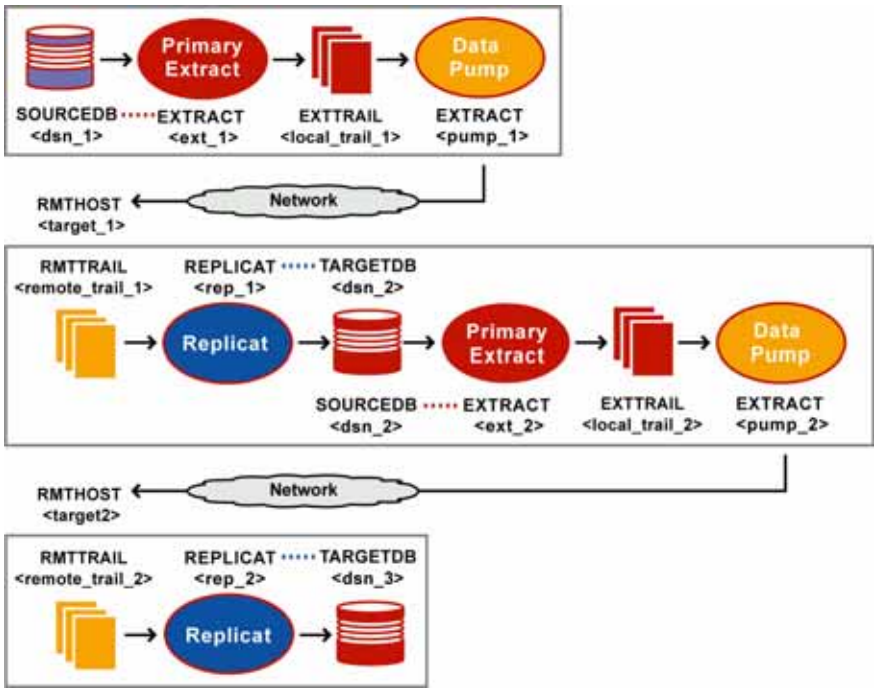
- One or more of the target systems does not have a direct connection to the source, but the intermediary system can connect in both directions.
- You want to limit network activity from the source system.
- You are sending data to two or more servers that are very far apart geographically, such as from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

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. You configure this Extract group to capture Replicat activity and to ignore local business application activity. The Extract parameters that control this behavior are IGNOREAPPLOPS and GETREPLICATES.
- 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** If you *do not* need to apply the replicated changes to a database on the secondary system, see “Creating a reporting configuration with a data pump on an intermediary system” on page 50.

Figure 11 Cascading configuration



Refer to Figure 11 for a visual representation of the objects you will be creating.

Source system

To configure the Manager process on the source

1. On the source, configure the Manager process according to the instructions in Chapter 2.
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

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

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

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

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

```
ADD EXTTRAIL <local_trail_1>, EXTRACT <ext>
```

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

5. 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.

```
-- Identify the Extract group:
EXTRACT <ext_1>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and specify encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_1>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

To configure the data pump on the source

6. 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.

7. 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.

8. 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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of second system in cascade
-- and optional encryption of data over TCP/IP:
RMTHOST <target_1>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the second system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_1>
-- Allow mapping, filtering, conversion or pass data through as-is.
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Second system in the cascade

To configure the Manager process on the second system

9. On the second system, configure the Manager process according to the instructions in Chapter 2.
10. 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

11. Create a Replicat checkpoint table. For instructions, see “Creating a checkpoint table” on page 185.
12. 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>, 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.

13. 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.

```
-- Identify the Replicat group:
REPLICAT <rep_1>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table> [, DEF <template name>];
```

To configure an Extract group on the second system

14. On the second system, use the ADD EXTRACT command to create a local Extract group. For documentation purposes, this group is called *ext_2*.

```
ADD EXTRACT <ext_2>, {TRANLOG | INTEGRATED TRANLOG}, BEGIN <time>
[, THREADS <n>]
```

- For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

15. On the second system, use the ADD RMTTRAIL command to specify a remote 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.

16. 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.

```
-- Identify the Extract group:
EXTRACT <ext_2>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_2>
-- Ignore local DML, capture Replicat DML:
IGNOREAPPLOPS, GETREPLICATES
-- Specify tables to be captured:
TABLE <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

17. 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.

18. 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.

19. 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>
-- Specify database login information if using NOPASSTHRU:
[SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of third system in cascade
-- and optional encryption of data over TCP/IP:
RMTHOST <target_2>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the third system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_2>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Third system in the cascade

To configure the Manager process

20. On the third system, configure the Manager process according to the instructions in Chapter 2.
21. In the Manager parameter file, use the PURGEOLDEXTRACTS parameter to control the purging of files from the trail.

To configure the Replicat group

22. On the third system, create a Replicat checkpoint table. For instructions, see "Creating a checkpoint table" on page 185.
23. 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>, EXTTRAIL <remote_trail_2>, BEGIN <time>
```

- Use the EXTTRAIL option to link the *rep_2* group to the *remote_trail_2* trail.

24. 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.

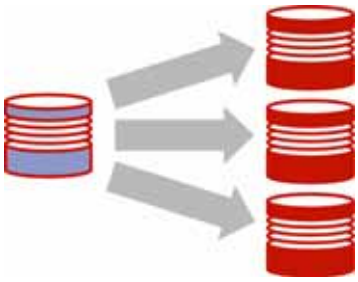
```
-- Identify the Replicat group:
REPLICAT <rep_2>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_3>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

CHAPTER 6

Using Oracle GoldenGate for real-time data distribution

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

Fault tolerance

For a data distribution configuration, the use of data pumps 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 a data-pump Extract group in the source configuration, one 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.

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:

- Native 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.

Data volume

The standard configuration is sufficient if:

- The transaction load is consistent and of moderate volume that is spread out more or less evenly among all of the objects to be replicated.

  and...

- There are none of the following: tables that are subject to long-running transactions, tables that have a very large number of columns that change, or tables that contain columns for which Oracle GoldenGate must fetch from the database (generally columns with LOBs, columns that are affected by SQL procedures executed by Oracle GoldenGate, and columns that are not logged to the transaction log).

If your environment does not satisfy those conditions, consider adding one or more sets of parallel processes. For more information, see the *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide*.

Read-only vs. high availability

This configuration supports read-only targets. If any target in this configuration will also be used for transactional activity in support of high availability, see "Using Oracle GoldenGate for active-active high availability" on page 93.

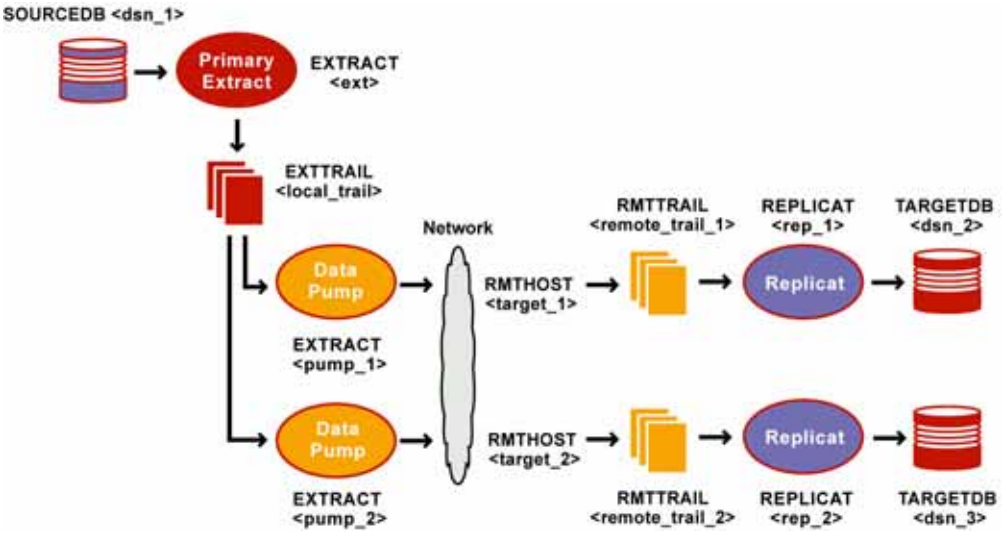
Additional information

- For additional system and database configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database type.
- For information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see "Configuring online change synchronization" on page 184.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- For more information about tuning this configuration, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

Creating a data distribution configuration

Refer to Figure 12 for a visual representation of the objects you will be creating.

Figure 12 Oracle GoldenGate configuration elements for data distribution



Source system

To configure the Manager process

1. On the source, configure the Manager process according to the instructions in Chapter 2.
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

3. On the source, use the ADD EXTRACT command to create a primary Extract group. For documentation purposes, this group is called *ext*.

   ```
   ADD EXTRACT <ext>, {TRANLOG | INTEGRATED TRANLOG}, BEGIN <time>,
   [, THREADS]
   ```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

4. 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.

5. 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.

```
-- Identify the Extract group:
EXTRACT <ext>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

   - Use EXTTRAIL to specify the local trail.

To configure the data pump Extract groups

6. 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.

7. 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.

8. 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.

Data pump_1

```
-- Identify the data pump group:
EXTRACT <pump_1>
-- Specify database login information if using NOPASSTHRU:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption options if input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the first target system
-- and optional encryption of data over TCP/IP:
RMTHOST <target_1>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify remote trail and encryption options on first target system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_1>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

Data pump_2

```
-- Identify the data pump group:
EXTRACT <pump_2>
-- Specify database login information if using NOPASSTHRU:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption options if input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the second target system
-- and optional encryption of data over TCP/IP:
RMTHOST <target_2>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify remote trail and encryption options on second target system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_2>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Target systems

To configure the Manager process

9. On each target, configure the Manager process according to the instructions in Chapter 2.

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

To configure the Replicat groups

11. On each target, create a Replicat checkpoint table. For instructions, see "Creating a checkpoint table" on page 185.

12. 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*.

    Target_1

    ```
    ADD REPLICAT <rep_1>, EXTTRAIL <remote_trail_1>, BEGIN <time>
    ```

    Target_2

    ```
    ADD REPLICAT <rep_2>, EXTTRAIL <remote_trail_2>, BEGIN <time>
    ```

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

13. 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.

    Target_1

    ```
    -- Identify the Replicat group:
    REPLICAT <rep_1>
    -- State whether or not source and target definitions are identical:
    SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
    -- Specify database login information as needed for the database:
    [TARGETDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
       [<encryption options>]]
    -- Specify decryption options if input trail is encrypted.
    DECRYPTTRAIL <encryption options>
    -- Specify error handling rules:
    REPERROR (<error>, <response>)
    -- Specify tables for delivery:
    MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
    ```

Target_2

```
-- Identify the Replicat group:
REPLICAT <rep_2>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_3>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption options if input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

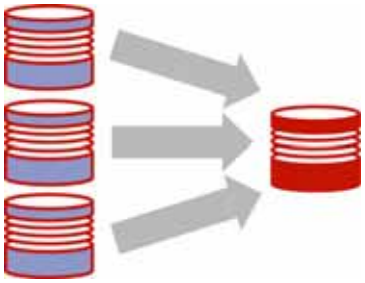
- You can use any number of MAP statements for any given Replicat group. All MAP statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

CHAPTER 7

Configuring Oracle GoldenGate for real-time data warehousing

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

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. For more information about resolving conflicts, see "Using Oracle GoldenGate for active-active high availability" on page 93.

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.

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:

- Native 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.

Data volume

The standard configuration is sufficient if:

- The transaction load is consistent and of moderate volume that is spread out more or less evenly among all of the objects to be replicated.

  and...

- There are none of the following: tables that are subject to long-running transactions, tables that have a very large number of columns that change, or tables that contain columns for which Oracle GoldenGate must fetch from the database (generally columns with LOBs, columns that are affected by SQL procedures executed by Oracle GoldenGate, and columns that are not logged to the transaction log).

If your environment does not satisfy those conditions, consider adding one or more sets of parallel processes. For more information, see the *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide*.

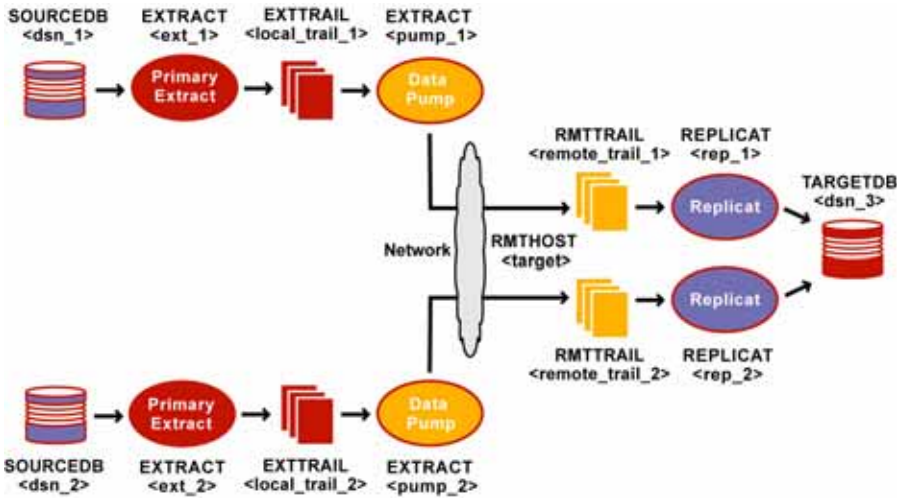
Additional information

- For additional system and database configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database type.
- For additional information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see “Configuring online change synchronization” on page 184.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- For more information about tuning this configuration, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

Creating a data warehousing configuration

Refer to Figure 13 for a visual representation of the objects you will be creating.

Figure 13 Configuration for data warehousing



Source systems

To configure the Manager process

1. On each source, configure the Manager process according to the instructions in Chapter 2.
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

3. 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*.

   Extract_1

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

   Extract_2

   ```
   ADD EXTRACT <ext_2>, (TRANLOG | INTEGRATED TRANLOG}, BEGIN <time>
   [, THREADS <n>]
   ```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

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

Extract_1

```
ADD EXTTRAIL <local_trail_1>, EXTRACT <ext_1>
```

Extract_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.

5. 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.

Extract_1

```
-- Identify the Extract group:
EXTRACT <ext_1>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_1>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

Extract_2

```
-- Identify the Extract group:
EXTRACT <ext_2>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

To configure the data pumps

6. 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*.

Data pump_1

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

Data 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 trail on the local system.

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

   Source_1

   ```
   ADD RMTTRAIL <remote_trail_1>, EXTRACT <pump_1>
   ```

   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.

8. 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.

   Data pump_1

   ```
   -- Identify the data pump group:
   EXTRACT <pump_1>
   -- Specify database login information as needed for the database:
   [SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
      [<encryption options>]]
   -- Specify decryption if the input trail is encrypted.
   DECRYPTTRAIL <encryption options>
   -- Specify the name or IP address of the target system
   -- and optional encryption of data over TCP/IP:
   RMTHOST <target>, MGRPORT <portnumber>, ENCRYPT <encryption options>
   -- Specify the remote trail and encryption options on the target system:
   ENCRYPTTRAIL <encryption options>
   RMTTRAIL <remote_trail_1>
   -- Allow mapping, filtering, conversion or pass data through as-is:
   [PASSTHRU | NOPASSTHRU]
   -- Specify tables to be captured:
   TABLE <owner>.<table>;
   ```

Data pump_2

```
-- Identify the data pump group:
EXTRACT <pump_2>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOST <target>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the target system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_2>
-- Allow mapping, filtering, conversion or pass data through as-is:
[PASSTHRU | NOPASSTHRU]
-- Specify tables to be captured:
TABLE <owner>.<table>;
```

- Use NOPASSTHRU if the data pump will be filtering or converting data, and also use the SOURCEDB and USERID parameters as appropriate for the database, to enable definitions lookups. If the data pump will not be filtering or converting data, use PASSTHRU to bypass the lookups.

NOTE To use PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. You can combine normal processing with pass-through processing by pairing PASSTHRU and NOPASSTHRU with different TABLE statements.

Target system

To configure the Manager process

9. Configure the Manager process according to the instructions in Chapter 2.
10. In the Manager parameter file, use the PURGEOLDEXTRACTS parameter to control the purging of files from the trail.

To configure the Replicat groups

11. 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*.

Replicat_1

```
ADD REPLICAT <rep_1>, EXTTRAIL <remote_trail_1>, BEGIN <time>
```

Replicat_2

```
ADD REPLICAT <rep_2>, EXTTRAIL <remote_trail_2>, BEGIN <time>
```

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

12. 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.

    Replicat_1

```
-- Identify the Replicat group:
REPLICAT <rep_1>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_3>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

    Replicat_2

```
-- Identify the Replicat group:
REPLICAT <rep_2>
-- State whether or not source and target definitions are identical:
SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
-- Specify database login information as needed for the database:
[TARGETDB <dsn_3>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted.
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

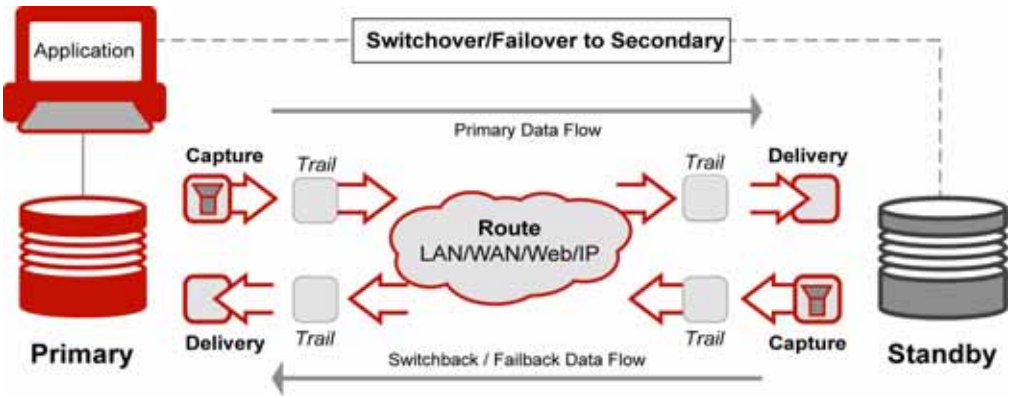
    - You can use any number of MAP statements for any given Replicat group. All MAP statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

CHAPTER 8

Using Oracle GoldenGate to maintain a live standby database

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

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 Chapter 9 on page 93.

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 were disabled during the setup procedures that were outlined in the Oracle GoldenGate *Installation and Setup Guide* 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 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 the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Data volume

The standard configuration is sufficient if:

- The transaction load is consistent and of moderate volume that is spread out more or less evenly among all of the objects to be replicated.

  and...

- There are none of the following: tables that are subject to long-running transactions, tables that have a very large number of columns that change, or tables that contain columns for which Oracle GoldenGate must fetch from the database (generally columns with LOBs, columns that are affected by SQL procedures executed by Oracle GoldenGate, and columns that are not logged to the transaction log).

If your environment does not satisfy those conditions, consider adding one or more sets of parallel processes. For more information, see the *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide*.

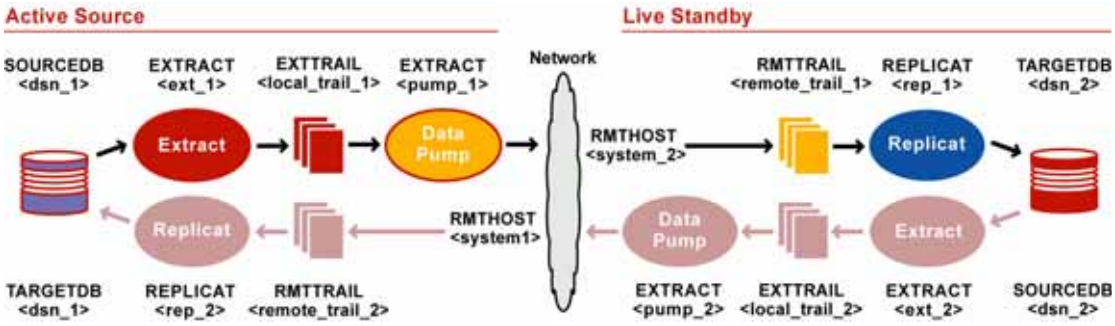
Additional information

- For additional system and database configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database type.
- For additional information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see “Configuring online change synchronization” on page 184.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- For more information about tuning this configuration, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

Creating a live standby configuration

Refer to Figure 14 for a visual representation of the objects you will be creating.

Figure 14 Oracle GoldenGate configuration elements for live standby



Prerequisites on both systems

1. Create a Replicat checkpoint table. For instructions, see "Creating a checkpoint table" on page 185.
2. Configure the Manager process according to the instructions in Chapter 2.

Configuration from active source to standby

To configure the primary Extract group

Perform these steps on the active source.

1. Use the ADD EXTRACT command to create a primary Extract group. For documentation purposes, this group is called *ext_1*.
2.

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

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

3. 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.

4. 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.

```
-- Identify the Extract group:
EXTRACT <ext_1>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_1>
-- Specify sequences to be captured:
SEQUENCE <owner.sequence>;
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

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.

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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the standby system
-- and optional encryption of data over TCP/IP:
RMTHOST <system_2>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the standby system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_1>
-- Pass data through without mapping, filtering, conversion:
PASSTHRU
-- Specify sequences to be captured:
SEQUENCE <owner.sequence>;
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

NOTE PASSTHRU mode is assumed because source and target data structures are usually identical in a live standby configuration. In this mode, no column mapping, filtering, SQLEXEC functions, transformation, or other data manipulation can be performed.

To configure the Replicat group

Perform these steps on the live standby system.

1. Create a Replicat checkpoint table. For instructions, see “Creating a checkpoint table” on page 185.
2. Use the ADD REPLICAT command to create a Replicat group. For documentation purposes, this group is called *rep_1*.

```
ADD REPLICAT <rep_1>, EXTTRAIL <remote_trail_1>, BEGIN <time>
```

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

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.

```
-- 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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.*, TARGET <owner>.*;
-- Exclude specific tables from delivery if needed:
MAPEXCLUDE <owner.table>
```

Configuration from standby to active source

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. 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>
[, THREADS <n>]
```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

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.

```
-- Identify the Extract group:
EXTRACT <ext_2>
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify local trail and encryption options that this Extract writes to:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_2>
-- Specify sequences to be captured:
SEQUENCE <owner.sequence>;
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

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 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.

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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the active source system
-- and optional encryption of data over TCP/IP:
RMTHOST <system_1>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify remote trail and encryption options on active source system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_2>
-- Pass data through without mapping, filtering, conversion:
PASSTHRU
-- Specify sequences to be captured:
SEQUENCE <owner.sequence>;
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

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>, EXTTRAIL <remote_trail_2>, BEGIN <time>
```

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

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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption if the input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Handle collisions between failback data copy and replication:
HANDLECOLLISIONS
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.*, TARGET <owner>.*;
-- Exclude specific tables from delivery if needed:
MAPEXCLUDE <owner.table>
```

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

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>
```

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

```
LAG 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

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

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.

To move users to the live standby

Perform these steps on the live standby system

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 the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled 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>
   ```

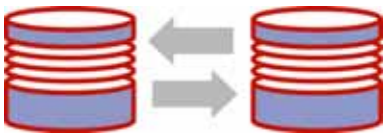
5. On the **primary** system, allow users to access the applications.

CHAPTER 9

Using Oracle GoldenGate for active-active high availability

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. Bidirectional synchronization is supported for all database types that are supported by Oracle GoldenGate.

Considerations for an active-active configuration

Supported databases

Oracle GoldenGate supports active-active configurations for:

- DB2 on z/OS and LUW
- MySQL
- Oracle
- SQL/MX
- SQL Server
- Sybase
- Teradata

Database-specific exclusions

Review the Oracle GoldenGate installation guide for your type of database to see if there are any limitations of support for a bi-directional 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.

DDL

Oracle GoldenGate supports DDL replication in an Oracle active-active configuration.

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.

Application design

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 key that you use for any given table

must;

- 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.

See the information in "Managing conflicts" on page 105 for additional options relating to columns that can be used in the WHERE clause for comparison purposes.

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. For certain Oracle database versions, you can use the DBOPTIONS parameter with the SUPPRESSTRIGGERS option to disable the triggers for the Replicat session. See the Oracle GoldenGate *Windows and UNIX Reference Guide* for important details.
- 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 IDENTITY columns cannot be used with bidirectional configurations for Sybase. See other IDENTITY limitations for SQL Server in the Oracle GoldenGate installation guide for that database.

Database-generated values

Do not replicate database-generated sequential values 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.

Data volume

The standard configuration is sufficient if:

- The transaction load is consistent and of moderate volume that is spread out more or less evenly among all of the objects to be replicated.

  and...

- There are none of the following: tables that are subject to long-running transactions, tables that have a very large number of columns that change, or tables that contain columns for which Oracle GoldenGate must fetch from the database (generally columns with LOBs, columns that are affected by SQL procedures executed by Oracle GoldenGate, and columns that are not logged to the transaction log).

If your environment does not satisfy those conditions, consider adding one or more sets of parallel processes. For more information, see the *Oracle GoldenGate Windows and UNIX Troubleshooting and Tuning Guide.*

Conflict resolution

Uniform conflict-resolution procedures must be in place on all systems to handle collisions that occur when modifications are made to identical sets of data on separate systems at (or almost at) the same time. In an active-active environment, 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.Oracle GoldenGate provides built-in conflict resolution support for insert, update, and delete conflicts. See "Managing conflicts" on page 105.

Additional information

- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see "Configuring online change synchronization" on page 184.
- For additional information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- For more information about tuning this configuration, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

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

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 capture of Replicat transactions (Teradata)

To prevent the capture of SQL that is applied by Replicat to a Teradata database, set the Replicat session to override Teradata replication. Use the following SQLEXEC statements at

the root level of the Replicat parameter file:

```
SQLEXEC "SET SESSION OVERRIDE REPLICATION ON;"
SQLEXEC "COMMIT;"
```

These SQLEXEC statements execute a procedure that sets the Replicat session automatically at startup.

Preventing capture of Replicat transactions (other databases)

To prevent the capture of SQL that is applied by Replicat to other database types, 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.

Before starting the Extract process, make certain that these parameters are either absent or set to GETAPPLOPS and IGNOREREPLICATES.

Identifying Replicat transactions

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

DB2 on z/OS and LUW

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

```
TRANLOGOPTIONS EXCLUDEUSER <user name>
```

This parameter statement marks all data 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 and NonStop SQL/MX

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 bi-directional configuration. FILTERTABLE identifies the name of the checkpoint table, so that Extract ignores transactions that contain any operations on it.

NOTE PURGEDATA is not supported for NonStop SQL/MX in a bidirectional configuration. Because PURGEDATA/TRUNCATE operations are DDL, they are implicit transactions, so Oracle GoldenGate cannot update the checkpoint table within that transaction.

Oracle

Do either of the following to specify the Replicat database user. All transactions generated by this user will be excluded from being captured. This information is available to Extract in the transaction record.

- Identify the Replicat database user by name with the following parameter statement in the Extract parameter file.

  ```
  TRANLOGOPTIONS EXCLUDEUSER <user name>
  ```

- Identify the Replicat database user by its numeric Oracle user-id (uid) with the following parameter statement in the Extract parameter file.

  ```
  TRANLOGOPTIONS EXCLUDEUSERID <user-id>
  ```

  NOTE Instead of using TRANLOGOPTIONS, you can create a trace table with the ADD TRACETABLE command in GGSCI. However, this method uses more processing overhead than the TRANLOGOPTIONS options do. For more information, see ADD TRACETABLE in the Oracle GoldenGate *Windows and UNIX Reference Guide*. Although not recommended, if a trace table and EXCLUDEUSER or EXCLUDEUSERID are both used for the same Extract configuration, they will both function correctly. Whatever transactions are specified for each one will be processed according to the rules of GETREPLICATES or IGNOREREPLICATES.

SQL Server

Identify the Replicat transaction name 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.

Sybase

Do any of the following:

- Identify a Replicat transaction name by using the following parameter statement in the Extract parameter file.

  ```
  TRANLOGOPTIONS EXCLUDETRANS <transaction name>
  ```

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

  ```
  TRANLOGOPTIONS EXCLUDEUSER <user name>
  ```

  EXCLUDEUSER marks all transactions generated by this user as Replicat transactions. The user name is included in the transaction record that is read by Extract.

- Do nothing and allow Replicat to use the default transaction name of ggs_repl.

Teradata

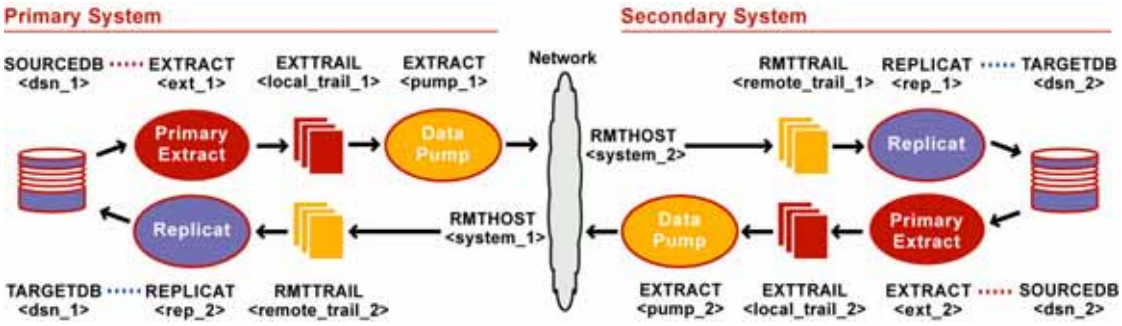
You do not need to identify Replicat transactions that are applied to a Teradata database.

Creating an active-active configuration

Refer to Figure 15 for a visual representation of the objects you will be creating.

NOTE To avoid conflicts, replication latency must be kept as low as possible. If this configuration does not provide acceptable latency levels, try adding parallel processes to it. See “Adding process groups” on page 261 for more information.

Figure 15 Oracle GoldenGate configuration elements for active-active synchronization



Prerequisites on both systems

1. Create a Replicat checkpoint table. For instructions, see “Creating a checkpoint table” on page 185.
2. Configure the Manager process according to the instructions in Chapter 2.

Configuration from primary system to secondary system

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>
   [, THREADS <n>]
   ```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

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.

   ```
   -- Identify the Extract group:
   EXTRACT <ext_1>
   ```

```
-- Specify database login information as needed for the database:
[SOURCEDB <dsn_1>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify the local trail that this Extract writes to
-- and encryption options:
ENCRYPTTRAIL <encryption options>
EXTTRAIL <local_trail_1>
-- Exclude Replicat transactions. Uncomment ONE of the following:
-- DB2 z/OS and LUW, Oracle, and Sybase:
-- TRANLOGOPTIONS EXCLUDEUSER <Replicat_user>
-- Oracle alternative to EXCLUDEUSER:
-- EXCLUDEUSERID <Oracle uid>
-- Oracle alternative to the TRANLOGOPTIONS options:
-- TRACETABLE <trace_table_name>
-- SQL Server and Sybase:
-- TRANLOGOPTIONS EXCLUDETRANS <transaction_name>
-- SQL/MX:
-- TRANLOGOPTIONS FILTERTABLE <checkpoint_table_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>)})
-- Specify tables to be captured and (optional) columns to fetch:
TABLE <owner>.* [, FETCHCOLS <cols> | FETCHCOLSEXCEPT <cols>];
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

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.

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>][, USERID <user>][, PASSWORD <pw>
   [<encryption options>]]
-- Specify decryption options if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the secondary system
-- and optional encryption of data over TCP/IP:
RMTHOST <system_2>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify remote trail and encryption options on secondary system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_1>
-- Pass data through without mapping, filtering, conversion:
PASSTHRU
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

NOTE Because data structures are usually identical in a bi-directional configuration, PASSTHRU mode improves performance.

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>, 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.

   ```
   -- 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>][, USERID <user>][, PASSWORD <pw>
      [<encryption options>]]
   ```

```
-- Specify decryption options if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery and  conflict-resolution routines:
MAP <owner>.*, TARGET <owner>.*,COMPARECOLS (ON <operation> {ALL | KEY |
KEYINCLUDING (<col list>) | ALLEXCLUDING (<col list>)}), RESOLVECONFLICT
(<conflict type> (<resolution name>, <resolution type> COLS
(<col>[,...]));
-- Exclude specific tables from delivery if needed:
MAPEXCLUDE <owner.table>
-- Specify mapping of exceptions to exceptions table:
MAP <owner>.*, TARGET <owner>.<exceptions>, EXCEPTIONSONLY;
```

Configuration from secondary system to primary 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 secondary system.

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>
   [, THREADS <n>]
   ```

   - For TRANLOG and INTEGRATED TRANLOG, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. INTEGRATED TRANLOG enabled integrated capture for an Oracle database.

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.

   ```
   -- Identify the Extract group:
   EXTRACT <ext_2>
   -- Specify database login information as needed for the database:
   [SOURCEDB <dsn_2>][, USERID <user>][, PASSWORD <pw>
      [<encryption options>]]
   -- Specify the local trail that this Extract writes to:
   EXTTRAIL <local_trail_2>
   -- Exclude Replicat transactions. Uncomment ONE of the following:
   -- DB2 z/OS and LUW, Oracle, and Sybase:
   -- TRANLOGOPTIONS EXCLUDEUSER <Replicat_user>
   -- Oracle alternative to EXCLUDEUSER:
   -- EXCLUDEUSERID <Oracle uid>
   -- Oracle alternative to the TRANLOGOPTIONS options:
   ```

```
-- TRACETABLE <trace_table_name>
-- SQL Server and Sybase:
-- TRANLOGOPTIONS EXCLUDETRANS <transaction_name>
-- SQL/MX:
-- TRANLOGOPTIONS FILTERTABLE <checkpoint_table_name>
-- Teradata:
-- SQLEXEC "SET SESSION OVERRIDE REPLICATION ON;"
-- SQLEXEC "COMMIT;"
-- Oracle:
-- TRACETABLE <trace_table_name>
-- 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 <owner>.* [, FETCHCOLS <cols> | FETCHCOLSEXCEPT <cols>];
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

NOTE To capture 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 the Oracle GoldenGate *Oracle Installation and Setup Guide*.

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>][, USERID <user>][, PASSWORD <pw>
      [<encryption options>]]
   ```

```
-- Specify decryption options if input trail is encrypted:
DECRYPTTRAIL <encryption options>
-- Specify the name or IP address of the primary system
-- and optional encryption of data over TCP/IP:
RMTHOST <system_1>, MGRPORT <portnumber>, ENCRYPT <encryption options>
-- Specify the remote trail and encryption options on the primary system:
ENCRYPTTRAIL <encryption options>
RMTTRAIL <remote_trail_2>
-- Pass data through without mapping, filtering, conversion:
PASSTHRU
-- Specify tables to be captured:
TABLE <owner>.*;
-- Exclude specific tables from capture if needed:
TABLEEXCLUDE <owner.table>
```

NOTE To propagate Oracle DBFS data, specify the internally generated local *read-write* DBFS tables in the TABLE statement on each node. For more information on configuring DBFS for propagation by Oracle GoldenGate, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.

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>, EXTTRAIL <remote_trail_2>, BEGIN <time>
   ```

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

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>][, USERID <user>][, PASSWORD <pw>
      [<encryption options>]]
   -- Specify decryption options if input trail is encrypted:
   DECRYPTTRAIL <encryption options>
   -- Specify error handling rules:
   REPERROR (<error>, <response>)
   -- Specify tables for delivery and  conflict-resolution routines:
   MAP <owner>.*, TARGET <owner>.*,COMPARECOLS (ON <operation> {ALL | KEY |
   KEYINCLUDING (<col list>) | ALLEXCLUDING (<col list>)}), RESOLVECONFLICT
   (<conflict type> (<resolution name>, <resolution type> COLS
   (<col>[,...]));
   -- Exclude specific tables from delivery if needed:
   MAPEXCLUDE <owner.table>
   -- Specify mapping of exceptions to exceptions table:
   MAP <owner>.*, TARGET <owner>.<exceptions>, EXCEPTIONSONLY;
   ```

NOTE To map Oracle DBFS data, map the internally generated source *read-write* tables to the remote *read-only* tables. For more information on configuring DBFS for propagation by Oracle GoldenGate, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.

Managing conflicts

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 the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide* for suggestions on improving the performance of the Oracle GoldenGate processes.

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.

Handling conflicts with 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 database must reside on a Windows, Linux, or UNIX system. It is not supported for databases on the NonStop platform.

CDR supports data types that can be compared with simple SQL and without explicit conversion:

- 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 or USEMAX option 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 the BATCHSQL parameter in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Configuring Oracle GoldenGate CDR

Follow these steps to configure the source database, target database, and Oracle GoldenGate for conflict detection and resolution. These steps are:

- Enabling the logging of before images
- Configuring the Oracle GoldenGate parameter files for conflict resolution
- Configuring the Oracle GoldenGate parameter files for error handling

Enabling the logging of before images

You may need to configure the source database to log the before image of any column on which conflict detection and resolution will be based for updates and deletes. For example, you might want to log the before image of a timestamp column or inventory amount.

- 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.
- For NonStop SQL/MX, create or alter the table to have the NOAUDITCOMPRESS attribute.
- For DB2 databases on all platforms supported by Oracle GoldenGate, use the GETUPDATEBEFORES, NOCOMPRESSUPDATES, and NOCOMPRESSDELETES parameters to ensure that the before image and the entire after image are available for CDR.

For detailed information about these parameters and commands, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. See the examples starting on page 112 for additional information on these parameters.

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 Extrat TABLE parameter to direct Extract to capture the before images that the database logs, and to write them to the trail. For DB2 databases on all platforms supported by Oracle GoldenGate, 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 confict 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. 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.

For detailed information about these parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. See the examples starting on page 112 for additional 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. 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 Oracle GoldenGate *Windows and UNIX Reference Guide* for details about these parameters.

> **NOTE** Conflict resolution is performed before the error handling performed by HANDLECOLLSIONS, INSERTMISSINGUPDATES, and REPERROR when those parameters are used.

2. (Optional) Create an exceptions table. When an exceptions table is used with an exceptions MAP statment (see step 3), 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 step 3). 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

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 "before" columns in the exceptions table, use the "before.<column>" syntax, which captures the before image from the trail record:

  ```
  COLMAP (USEDEFAULTS, <exceptions_col1> = before.<source_col1>, &
  <exceptions_col2> = before.<source_col2>, [...])
  ```

- To map the current image of the target row to "current" 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:

  ```
  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" on page 110 for how to combine these features in a COLMAP in the exceptions MAP to populate a detailed exceptions table.

See the Oracle GoldenGate *Windows and UNIX Reference Guide* for detailed help with usage and syntax for the parameters and column-conversion functions shown.

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 on page 112. 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.

```
-- REPERROR error handling: DEFAULT represents all error types. DISCARD
-- writes operations that could not be processed to a discard file.
```

```
REPERROR (DEFAULT, DISCARD)
   -- Specifies the discard file.
DISCARDFILE ./dirrpt/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 REPERROR.
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 on page 112. 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.

```
   -- REPERROR error handling: DEFAULT represents all error types. DISCARD
   -- writes operations that could not be processed to a discard file.
REPERROR (DEFAULT, DISCARD)
   -- Specifies the discard file.
DISCARDFILE ./dirrpt/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 REPERROR.
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 = :p1", PARAMS(p1 = 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.
dberrmsge = @GETENV("LASTERR", "DBERRMSG"), &
   -- captures and maps the name of the target table
tabname = @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:
-- 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 additional information about creating an exceptions table and using exceptions mapping, see "Handling Replicat errors during DML operations" on page 168.

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

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 DELROWEXISTS conflicts        1
   CDR DELROWMISSING conflicts       1
```

You can also view CDR statistics from GGSCI by using the STATS REPLICAT command with the REPORTCDR option:

```
STATS REPLICAT <group>, REPORTCDR
```

CDR example 1: All conflict types with USEMAX, OVERWRITE, DISCARD

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

- 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.
- 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.

Error handling

For an example of error handling to an exceptions table, see “Configuring the Oracle GoldenGate parameter files for error handling” on page 107.

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 3 INSERTROWEXISTS conflict with USEMAX resolution

Image	SQL	Comments
Before image in trail	`None (row was inserted on the source).`	

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 4 UPDATEROWEXISTS conflict with USEMAX resolution

Image	SQL	Comments

Before 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 before image of the resolution column.

After image in trail	`phone='222222'`
	`address='Holly'`
	`last_mod_time='9/1/10 5:00'`
		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	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 6:00'`
		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	SQL bind variables:
	`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'`
		This SQL returns a no-data-found error because the values for the balance, comment, and last_mod_time are different in the target.
		All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL.

Table 4 UPDATEROWEXISTS conflict with USEMAX resolution (continued)

Image	SQL	Comments
UPDATE applied by Replicat to resolve the conflict	SQL bind variables: 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'	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.

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 5 UPDATEROWMISSING conflict with OVERWRITE resolution

Image	SQL	Comments
Before image in trail	name='Jane' phone='333' address='Oracle Pkwy' salary=200 balance=200 comment=NULL last_mod_time='9/1/10 7:00'	
After image in trail	phone='4444' address='Holly' last_mod_time='9/1/10 8:00'	
Target database image	None (row for Jane is missing)	

Table 5 UPDATEROWMISSING conflict with OVERWRITE resolution (continued)

Image	SQL	Comments
Initial UPDATE applied by Replicat that detects the conflict	SQL bind variables: 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'	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	SQL bind variables: 1)'Jane' 2)'4444' 3)'Holly' 4)200 5)200 6)NULL 7)'9/1/10 8:00'	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 6 DELETEROWMISSING conflict with DISCARD resolution

Image	SQL	Comments
Before image in trail	name='Jane' phone='4444' address='Holly' salary=200 balance=200 comment=NULL last_mod_time='9/1/10 8:00'	
After image in trail	None	
Target database image	None (row missing)	

Table 6 DELETEROWMISSING conflict with DISCARD resolution (continued)

Image	SQL	Comments
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 7 DELETEROWEXISTS conflict with OVERWRITE resolution

Image	SQL	Comments
Before image in trail	name='Mary' phone='222222' address='Holly' salary=100 balance=100 comment=NULL last_mod_time='9/1/10 5:00'	
After image in trail	None	
Target database image	name='Mary' phone='1234567890' address='Oracle Pkwy' salary=100 balance=600 comment=com last_mod_time='9/1/10 7:00'	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.

Table 7 DELETEROWEXISTS conflict with OVERWRITE resolution (continued)

Image	SQL	Comments
Initial DELETE applied by Replicat that detects the conflict	SQL bind variables: 1)'Mary' 2)'222222' 3)'Holly' 4)100 5)100d 6)NULL 7)'9/1/10 5:00'	All columns are used in the WHERE clause because the COMPARECOLS statement is set to ALL. A no-data-found error occurs because of the difference between the before and current values.
DELETE applied by Replicat to resolve the conflict	SQL bind variables: 1)'Mary'	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

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

- 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.)

Error handling

For an example of error handling to an exceptions table, see "Configuring the Oracle GoldenGate parameter files for error handling" on page 107.

Table 8 UPDATEROWEXISTS with USEDELTA and USEMAX

Image	SQL	Comments

Before 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 before image of the resolution column for the USEMAX resolution.
		salary=100 is the before image for the USEDELTA resolution.

After image in trail	`phone='222222'`
	`address='Holly'`
	`salary=200`
	`comment='new'`
	`last_mod_time='9/1/10 5:00'`
		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.

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=600`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 4:00'`
		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.
		salary=600 is the current image of the target column for the USEDELTA resolution.

Table 8 UPDATEROWEXISTS with USEDELTA and USEMAX

Image	SQL	Comments
Initial UPDATE applied by Replicat that detects the conflict	SQL bind variables: 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'	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.)
UPDATE applied by Replicat to resolve the conflict for salary, using USEDELTA.	SQL bind variables: 1)200 2)100 3)'Mary'	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. 600 + (200 - 100) = 700
UPDATE applied by Replicat to resolve the conflict for the default columns, using USEMAX.	SQL bind variables: 1)'222222' 2)'Holly' 3)'new' 4)'9/1/10 5:00' 5)'Mary' 6)'9/1/10 5:00'	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. Note that the salary column is not set here, because it is resolved with the UPDATE from the USEDELTA resolution.

CRD example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

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

- 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.

Error handling

For an example of error handling to an exceptions table, see "Configuring the Oracle GoldenGate parameter files for error handling" on page 107.

Table 9 UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments

Before 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 before image of the resolution column for the USEMAX resolution.
		salary=100 and balance=100 are the before images for the USEDELTA resolution.

After image in trail	`phone='222222'`
	`address='Holly'`
	`salary=200`
	`comment='new'`
	`last_mod_time='9/1/10 5:00'`
		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.
		salary=200 is the only after image available for the USEDELTA resolution. For balance, the before image will be used in the calculation.

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Ralston'`
	`salary=600`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 4:00'`
		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.
		salary=600 and balance=600 are the current images of the target columns for USEDELTA.

Initial UPDATE applied by Replicat that detects the conflict	SQL bind variables:
	`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'`
		This SQL returns a no-data-found error because the values for the address, salary, balance and last_mod_time columns are different.

Table 9 UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments
UPDATE applied by Replicat to resolve the conflict for salary, using USEDELTA.	SQL bind variables: 1)200 2)100 3)'Mary'	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.
UPDATE applied by Replicat to resolve the conflict for USEMAX.	SQL bind variables: 1)'Holly' 2)'9/1/10 5:00' 3)'Mary' 4)'9/1/10 5:00'	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. Note that the salary column is not set here, because it is resolved with the UPDATE from the USEDELTA resolution.
UPDATE applied by Replicat for IGNORE.	SQL bind variables: 1)'222222' 2)'new' 3)'Mary'	IGNORE is specified for the DEFAULT column group (phone and comment), so no resolution SQL is applied.

CHAPTER 10

Configuring Oracle GoldenGate security

Overview of Oracle GoldenGate security options

You can use the following security features to protect your Oracle GoldenGate environment and the data that is being processed

Table 10 Oracle GoldenGate security options[1]

Security feature	What it secures	Description
Trail File Encryption page 126	Records written to trail files or an extract file across data links and within the files.	Use any of the following: ◆ 256-key byte substitution[2] ◆ Any Advanced Encryption Security (AES)[3] cipher: AES-128 AES-192 AES-256
Password Encryption page 130	Passwords used by Oracle GoldenGate components to log into the database. The clear-text password must be explicitly encrypted through GGSCI.	Use any of the following: ◆ AES-128 ◆ AES-192 ◆ AES-256 ◆ Blowfish[4]
TCP/IP Encryption page 133	Data sent across TCP/IP. On the target, Oracle GoldenGate decrypts the data before writing it to a trail, unless trail encryption also is specified.	Use any of the following: ◆ AES-128 ◆ AES-192 ◆ AES-256 ◆ Blowfish
Command Authentication page 136	Oracle GoldenGate commands issued through GGSCI.	Configure a CMDSEC (Command Security) file.

Table 10 Oracle GoldenGate security options[1]

Security feature	What it secures	Description
Trusted Connection page 138	TCP/IP connection to hosts that are outside a firewall.	Use any of the following: ◆ AES-128 ◆ AES-192 ◆ AES-256 ◆ Blowfish

[1] For additional guidelines on securing your data environment, see the Oracle Security Guide.

[2] Byte substitution provides a simple substitution of each byte of plain text with a different ciphertext value.

[3] Advanced Encryption Standard (AES) is a symmetric-key encryption standard that is used by governments and other organizations that require a high degree of data security. It offers three 128-bit block-ciphers: a 128-bit key cipher, a 192-bit key cipher, and a 256-bit key cipher. To use AES for any database other than Oracle, the path to the lib sub-directory of the Oracle GoldenGate installation directory must be set with the LD_LIBRARY_PATH or SHLIB_PATH variable (UNIX) or the PATH variable (Windows).

[4] Blowfish encryption: A keyed symmetric-block cipher. The Oracle GoldenGate implementation of Blowfish has a 64-bit block size with a variable-length key size from 32 bits to 128 bits.

Encrypting a trail or file

This section shows how to specify encryption settings for trails or extract files that store data that is being processed by Oracle GoldenGate. For correct placement in parameter files, see the example on page 127.

Extract parameter file

Add an ENCRYPTTRAIL parameter statement before the EXTTRAIL, RMTTRAIL, EXTFILE, or RMTFILE parameter that names the trail or file that this Extract writes to.

```
ENCRYPTTRAIL [{AES128 | AES192 | AES256} KEYNAME <keyname>]
```

Where:

- AES128 encrypts with AES-128.
- AES192 encrypts with AES-192.
- AES256 encrypts with AES-256.
- KEYNAME <keyname> specifies the logical look-up name of an encryption key in the ENCKEYS file. See "Generating encryption keys" on page 134. KEYNAME is required for AES options.
- ENCRYPTTRAIL by itself (without any input arguments) encrypts files with 256-key byte substitution. This is insecure and should not be used in a production environment. Use only for backward compatibility with older Oracle GoldenGate versions.

NOTE Although not typical, it is possible for one Extract to write to multiple trail files. In that configuration, you can use one ENCRYPTTRAIL statement to encrypt all of them. If any of the trails or files is not to be encrypted, place the associated EXTTRAIL, RMTTRAIL, EXTFILE, or RMTFILE statement *before* the ENCRYPTTRAIL statement.

Data pump Extract parameter file

1. To decrypt a trail or file that a data pump reads, so that the pump can process it, add a DECRYPTTRAIL parameter statement. The decryption algorithm and key must match the ones that were used to encrypt the trail.

   ```
   DECRYPTTRAIL [{AES128 | AES192 | AES256} KEYNAME <keyname>]
   ```

   Where:

   - Use DECRYPTTRAIL without options if ENCRYPTTRAIL was used without options.
   - Use AES128, AES192, or AES256 if a matching AES cipher was used in ENCRYPTTRAIL.
   - Use KEYNAME <keyname> if AES is being used, and specify the same logical look-up name that is used on the source.

2. To encrypt a trail or file that the data pump writes to, add an ENCRYPTTRAIL statement before the EXTTRAIL, RMTTRAIL, EXTFILE, or RMTFILE parameter statement. Place this statement after the DECRYPTTRAIL statement.

   NOTE ENCRYPTTRAIL cannot be used when FORMATASCII is used to write data to a file in ASCII format; the trail or file must be written in the default canonical format. In addition, ENCRYPTTRAIL set to an AES option cannot be used with the ETOLDFORMAT parameter; only 256-key byte substitution is supported with ETOLDFORMAT.

Replicat parameter file

Add a DECRYPTTRAIL parameter statement if the trail that Replicat reads is encrypted. The decryption algorithm and key must match the ones that were used to encrypt the trail.

Example parameter files: all trails encrypted

This example shows how to use encryption options for different trails across a simple configuration that uses one primary Extract, one data pump, and one Replicat. It assumes that the primary Extract and the data pump write to one trail each. In these examples, note that the Oracle GoldenGate database user's password is encrypted with password encryption (see page 130). See also "Example parameter files: one trail encrypted, one trail non-encrypted" on page 128.

NOTE These examples are basic parameter files that highlight the encryption options and do not necessarily represent all of the parameters that should be used in any given Oracle GoldenGate configuration.

Extract parameter file

```
EXTRACT capt
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
DISCARDFILE /ogg/capt.dsc, PURGE
-- Encrypt the output trail. Use AES-192 with encryption key mykey1.
ENCRYPTTRAIL AES192 KEYNAME mykey1
EXTTRAIL /ogg/dirdat/aa
TABLE FIN.*;
TABLE SALES.*;
```

Data pump parameter file

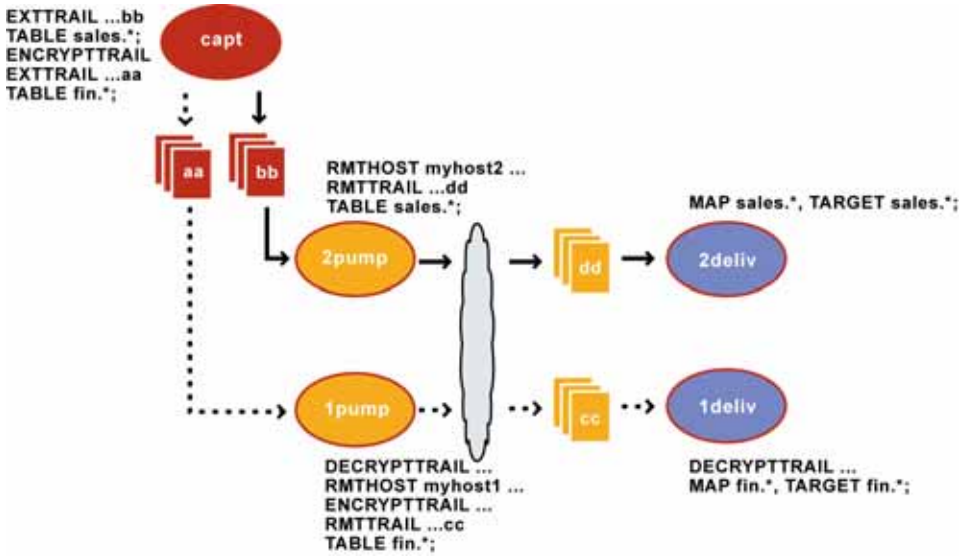
```
EXTRACT pump
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
DISCARDFILE /ogg/pmp.dsc, PURGE
-- Decrypt the input trail. Use encryption key mykey1.
DECRYPTTRAIL AES192 KEYNAME mykey1
-- Encrypt the output trail. Use AES-192 and encryption key mykey2.
RMTHOST myhost1, MGRPORT 7809
ENCRYPTTRAIL AES192 KEYNAME mykey2
RMTTRAIL /ogg/dirdat/bb
TABLE FIN.*;
TABLE SALES.*;
```

Replicat parameter file

```
REPLICAT deliv
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
ASSUMETARGETDEFS
DISCARDFILE /ogg/deliv.dsc, PURGE
-- Decrypt the input trail. Use encryption key mykey2.
DECRYPTTRAIL AES192 KEYNAME mykey2
MAP FIN.*, TARGET FIN.*;
MAP SALES.*, TARGET SALES.*;
```

Example parameter files: one trail encrypted, one trail non-encrypted

This example shows how to turn encryption on and off for different trails or files that are written by the same Extract or data pump process. In this type of configuration, any EXTTRAIL, RMTTRAIL, EXTFILE, or RMTFILE parameters that specify trails or files that are *not* to be encrypted must precede the ENCRYPTTRAIL statement, which then turns on encryption for any subsequently listed trails or files. In these examples, note that the Oracle GoldenGate database user's password is encrypted with password encryption (see page 130).



Extract parameter file

This Extract process writes to two local trails, one of which is encrypted.

```
EXTRACT capt
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
DISCARDFILE /ogg/capt.dsc, PURGE
-- Do not encrypt this trail.
EXTTRAIL /ogg/dirdat/bb
TABLE SALES.*;
-- Encrypt this trail with AES-192.
ENCRYPTTRAIL AES192 KEYNAME mykey1
EXTTRAIL /ogg/dirdat/aa
TABLE FIN.*;
```

Data pump 1 parameter file

This pump reads encrypted local trail /ogg/dirdat/aa that contains data from tables owned by fin and sends the data to remote trail /ogg/dirdat/cc on host myhost1.

```
EXTRACT 1pump
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
DISCARDFILE /ogg/1pmp.dsc, PURGE
-- Decrypt the trail this pump reads. Use encryption key mykey1.
DECRYPTTRAIL AES192 KEYNAME mykey1
-- Encrypt the trail this pump writes to, using AES-192.
RMTHOST myhost1, MGRPORT 7809
ENCRYPTTRAIL AES192 KEYNAME mykey2
RMTTRAIL /ogg/dirdat/cc
TABLE FIN.*;
```

Data pump 2 parameter file

This pump reads non-encrypted local trail /ogg/dirdat/bb that contains data from tables owned by sales and sends the data to remote trail /ogg/dirdat/dd on host myhost2.

```
EXTRACT 2pump
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
DISCARDFILE /ogg/2pmp.dsc, PURGE
RMTHOST myhost2, MGRPORT 7809
RMTTRAIL /ogg/dirdat/dd
TABLE SALES.*;
```

Replicat1 (on myhost1) parameter file

This Replicat reads encrypted trail /ogg/dirdat/cc that contains encrypted data from tables owned by fin and applies it to the target database on myhost1.

```
REPLICAT 1deliv
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey2
ASSUMETARGETDEFS
DISCARDFILE /ogg/1deliv.dsc, PURGE
-- Decrypt the trail this Replicat reads. Use encryption key mykey2.
DECRYPTTRAIL AES192 KEYNAME mykey2
MAP FIN.*, TARGET FIN.*;
```

Replicat 2 (on myhost2) parameter file

This Replicat reads non-encrypted trail /ogg/dirdat/dd that contains data from tables owned by sales and applies it to the target database on myhost2.

```
REPLICAT 2deliv
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey2
ASSUMETARGETDEFS
DISCARDFILE /ogg/2deliv.dsc, PURGE
MAP SALES.*, TARGET SALES.*;
```

Encrypting a database user password

This section shows how to a database password that is used by any Oracle GoldenGate process to log into the database management system. Using an encrypted password involves two steps:

- Encrypt the password
- Specify the encrypted password in a parameter or command

To encrypt the password

1. Run GGSCI.
2. Issue the ENCRYPT PASSWORD command.

```
ENCRYPT PASSWORD <password> <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}
```

Where:

- <password> is the clear-text login password. Do not enclose the password within quotes. If the password is case-sensitive, type it that way.
- <algorithm> specifies the encryption algorithm to use:
  - AES128 uses the AES-128 cipher, which has a key size of 128 bits.
  - AES192 uses the AES-192 cipher, which has a key size of 192 bits.
  - AES256 uses the AES-256 cipher, which has a key size of 256 bits.
  - BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use BLOWFISH only for backward compatibility with earlier Oracle GoldenGate versions.
- ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. The key name is used to look up the actual key in the ENCKEYS file. Using a user-defined key and an ENCKEYS file is required for AES encryption. To create a key and ENCKEYS file, see "Generating encryption keys" on page 134.
- ENCRYPTKEY DEFAULT directs Oracle GoldenGate to generate a random key that is then stored in the trail so that decryption can be performed by the downstream process. This type of key is insecure and should not be used in a production environment. Use this option only when BLOWFISH is specified. ENCRYPT PASSWORD returns an error if AES is used with DEFAULT.

If no algorithm is specified, AES128 is the default for all database types except DB2 on z/OS and NonStop SQL/MX, where BLOWFISH is the default.

3. The encrypted password is output to the screen when you run the ENCRYPT PASSWORD command.

Examples of ENCRYPT PASSWORD

```
ENCRYPT PASSWORD mypassword AES256 ENCRYPTKEY mykey1
ENCRYPT PASSWORD mypassword BLOWFISH ENCRYPTKEY mykey1
ENCRYPT PASSWORD mypassword BLOWFISH ENCRYPTKEY DEFAULT
```

To specify the encrypted password in a parameter or command

1. Copy the encrypted password that you generated with the ENCRYPT PASSWORD command (page 130), and then paste it into the appropriate Oracle GoldenGate parameter statement or command as shown in Table 11. Option descriptions follow the table.

Table 11 Specifying encrypted passwords in the Oracle GoldenGate parameter file

To encrypt a password for...	Use this parameter...
Oracle GoldenGate database user[1]	`USERID <user>, PASSWORD <password>, & <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`

Table 11 Specifying encrypted passwords in the Oracle GoldenGate parameter file

To encrypt a password for...	Use this parameter...
Oracle GoldenGate user in Oracle ASM instance	`TRANLOGOPTIONS ASMUSER SYS@<ASM_instance_name>, & ASMPASSWORD <password>, & <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`
Password substitution for {CREATE | ALTER} USER <name> IDENTIFIED BY <password>	`DDLOPTIONS DEFAULTUSERPASSWORD <password> <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`
Oracle TDE shared-secret password	`DBOPTIONS DECRYPTPASSWORD <password>`[2] `<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`
Oracle GoldenGate database login from the GGSCI command interface	`DBLOGIN USERID <user>, PASSWORD <password>, <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`

[1] Syntax elements required for USERID vary by database type. See the Oracle GoldenGate reference documentation for more information.
[2] This is the shared secret.

Where:

- <user> is the database user name for the Oracle GoldenGate process or (Oracle only) a host string. For Oracle ASM, the user must be SYS.
- < password> is the encrypted password that is copied from the ENCRYPT PASSWORD command results.
- <algorithm> specifies the encryption algorithm that was used to encrypt the password: AES128, AES192, AES256, or BLOWFISH. AES128 is the default if the default key is used and no algorithm is specified.
- ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option.
- ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.

Examples of how to use an encrypted password in parameters and commands

```
SOURCEDB db1 USERID ogg,&
PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
BLOWFISH, ENCRYPTKEY securekey1

USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
BLOWFISH, ENCRYPTKEY DEFAULT

TRANLOGOPTIONS ASMUSER SYS@asm1, &
ASMPASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

DBLOGIN USERID ogg, PASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

DDLOPTIONS DEFAULTUSERPASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey

DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey

DDLOPTIONS PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey
```

Encrypting data sent across TCP/IP

This section shows how to encrypt the data that Extract or a data pump sends across TCP/IP connections to a remote target. On the target, Oracle GoldenGate decrypts the data before writing it to a trail, unless trail encryption also is specified.

1. On the source system, generate one or more encryption keys and create an ENCKEYS file. See "Generating encryption keys" on page 134.
2. Copy the finished ENCKEYS file to the Oracle GoldenGate installation directory on all target systems. The key names and values in the source ENCKEYS file must match those in each of the target ENCKEYS files, or else the data exchange will fail and Extract and Collector will abort with a message similar to the following:

   ```
   GGS error 118 - TCP/IP Server with invalid data.
   ```

3. Depending on whether this is a regular Extract group or a passive Extract group (see page 138), use the ENCRYPT option of either the RMTHOST or RMTHOSTOPTIONS parameter to specify the encryption algorithm and the logical key name as shown:

   ```
   RMTHOST <host>, MGRPORT <port>,
   ENCRYPT <algorithm> KEYNAME <keyname>

   RMTHOSTOPTIONS ENCRYPT <algorithm> KEYNAME <keyname>
   ```

Where:

- <algorithm> specifies the encryption algorithm to use:
  - AES128 uses the AES-128 cipher, which has a key size of 128 bits.
  - AES192 uses the AES-192 cipher, which has a key size of 192 bits.
  - AES256 uses the AES-256 cipher, which has a key size of 256 bits.
  - BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use BLOWFISH only for backward compatibility with earlier Oracle GoldenGate versions.
- KEYNAME <keyname> specifies the logical name of an encryption key in the ENCKEYS lookup file. The key name is used to look up the actual key in the ENCKEYS file.

Examples of how to configure TCP/IP encryption

```
RMTHOST myhost, MGRPORT 7840, ENCRYPT BLOWFISH, KEYNAME mykey

RMTHOST myhost, MGRPORT 7840, ENCRYPT AES256, KEYNAME mykey

RMTHOSTOPTIONS ENCRYPT AES192, KEYNAME mykey
```

4. If using a static Collector, append the following additional parameters in the Collector startup string:

```
-KEYNAME <keyname>
-ENCRYPT <algorithm>
```

Collector matches these parameters to those specified with the KEYNAME and ENCRYPT options of RMTHOST.

Generating encryption keys

You must generate and store encryption keys when using:

- ENCRYPTTRAIL with KEYNAME <keyname> (see page 126)
- ENCRYPT PASSWORD with ENCRYPTKEY <keyname> (see page 130)
- RMTHOST or RMTHOSTOPTIONS with ENCRYPT (see page 133)

This procedure is *not required* if you are using the following encryption options:

- ENCRYPT PASSWORD with ENCRYPTKEY DEFAULT (valid only when using BLOWFISH)
- ENCRYPTTRAIL without options (for 256-key byte substitution)

In this procedure you will:

- Create one or more encryption keys.
- Store the keys in an ENCKEYS lookup file on the source system.
- Copy the ENCKEYS file to each target system.

You can define your own key or run the Oracle GoldenGate KEYGEN utility to create a random key.

To define your own key

Use a tool of your choice. The key value can be up to 128 bits (16 bytes) as either of the following:

- a quoted alphanumeric string (for example "Dailykey")
- a hex string with the prefix 0x (for example 0x420E61BE7002D63560929CCA17A4E1FB)

To use KEYGEN to generate a key

Change directories to the Oracle GoldenGate home directory on the source system, and issue the following shell command. You can create multiple keys, if needed. The key values are returned to your screen. You can copy and paste them into the ENCKEYS file.

```
KEYGEN <key length> <n>
```

Where:

- <key length> is the encryption key length, up to 128 bits (16 bytes).
- <n> represents the number of keys to generate.

Example:

```
KEYGEN 128 4
```

To store the keys in an ENCKEYS lookup file

1. On the source system, open a new ASCII text file.
2. For each key value that you generated, enter a logical name of your choosing, followed by the key value itself.
   - The key name can be a string of 1 to 24 alphanumeric characters without spaces or quotes.
   - Place multiple key definitions on separate lines.
   - Do not enclose a key name or value within quotes; otherwise it will be interpreted as text.

   Use the following sample ENCKEYS file as a guide.

```
## Encryption keys

## Key name      Key value
superkey         0x420E61BE7002D63560929CCA17A4E1FB
secretkey        0x027742185BBF232D7C664A5E1A76B040
superkey1        0x42DACD1B0E94539763C6699D3AE8E200
superkey2        0x0343AD757A50A08E7F9A17313DBAB045
superkey3        0x43AC8DCE660CED861B6DC4C6408C7E8A
```

3. Save the file as the name ENCKEYS in all upper case letters, *without an extension*, in the Oracle GoldenGate installation directory.

4. Copy the ENCKEYS file to the target Oracle GoldenGate installation directory. The key names and values in the source ENCKEYS file must match those of the target ENCKEYS file, or else the data exchange will fail and Extract and Collector will abort with the following message:

   ```
   GGS error 118 – TCP/IP Server with invalid data.
   ```

Configuring command security

You can establish command security for Oracle GoldenGate to control which users have access to which Oracle GoldenGate functions. For example, you can allow certain users to issue INFO and STATUS commands, while preventing their use of START and STOP commands. Security levels are defined by the operating system's user groups.

To implement security for Oracle GoldenGate commands, you create a CMDSEC file in the Oracle GoldenGate directory. Without this file, access to all Oracle GoldenGate commands is granted to all users.

To implement command security

1. Open a new ASCII text file.
2. Referring to the following syntax and the example on page 137, create one or more security rules for each command that you want to restrict, one rule per line. List the rules in order from the most specific (those with no wildcards) to the least specific. Security rules are processed from the top of the CMDSEC file downward. The first rule satisfied is the one that determines whether or not access is allowed.

   Separate each of the following components with spaces or tabs.

   ```
   <command name> <command object> <OS group> <OS user> <YES | NO>
   ```

   Where:

   - <command name> is a GGSCI command name or a wildcard, for example START or STOP or *.
   - <command object> is any GGSCI command object or a wildcard, for example EXTRACT or REPLICAT or MANAGER.
   - <OS group> is the name of a Windows or UNIX user group. On a UNIX system, you can specify a numeric group ID instead of the group name. You can use a wildcard to specify all groups.
   - <OS user> is the name of a Windows or UNIX user. On a UNIX system, you can specify a numeric user ID instead of the user name. You can use a wildcard to specify all users.
   - <YES | NO> specifies whether access to the command is granted or prohibited.

3. Save the file as CMDSEC (using upper case letters on a UNIX system) in the Oracle GoldenGate home directory.

The following example illustrates the correct implementation of a CMDSEC file on a UNIX system.

Table 12 Sample CMDSEC file with explanations

File Contents	Explanation
`#GG command security`	Comment line
`STATUS REPLICAT * Smith NO`	STATUS REPLICAT is denied to user Smith.
`STATUS * dpt1 * YES`	Except for the preceding rule, all users in dpt1 are granted all STATUS commands.
`START REPLICAT root * YES`	START REPLICAT is granted to all members of the root group.
`START REPLICAT * * NO`	Except for the preceding rule, START REPLICAT is denied to all users.
`* EXTRACT 200 * NO`	All EXTRACT commands are denied to all groups with ID of 200.
`* * root root YES`	Grants the root user any command.
`* * * * NO`	Denies all commands to all users. This line covers security for any other users that were not explicitly granted or denied access by preceding rules. Without it, all commands would be granted to all users except for preceding explicit grants or denials.

The following *incorrect* example illustrates what to avoid when creating a CMDSEC file.

Table 13 Incorrect CMDSEC entries

File Contents	Description
`STOP * dpt2 * NO`	All STOP commands are denied to everyone in group dpt2.
`STOP * * Chen YES`	All STOP commands are granted to Chen.

The order of the entries in Table 13 causes a logical error. The first rule (line 1) denies all STOP commands to all members of group dpt2. The second rule (line 2) grants all STOP commands to user Chen. However, because Chen is a member of the dpt2 group, he has been denied access to all STOP commands by the second rule, even though he is supposed to have permission to issue them.

The proper way to configure this security rule is to set the user-specific rule before the more general rule(s). Thus, to correct the error, you would reverse the order of the two STOP rules.

Securing the CMDSEC File

Because the CMDSEC file is a source of security, it must be secured. You can grant read access as needed, but Oracle GoldenGate recommends denying write and delete access to

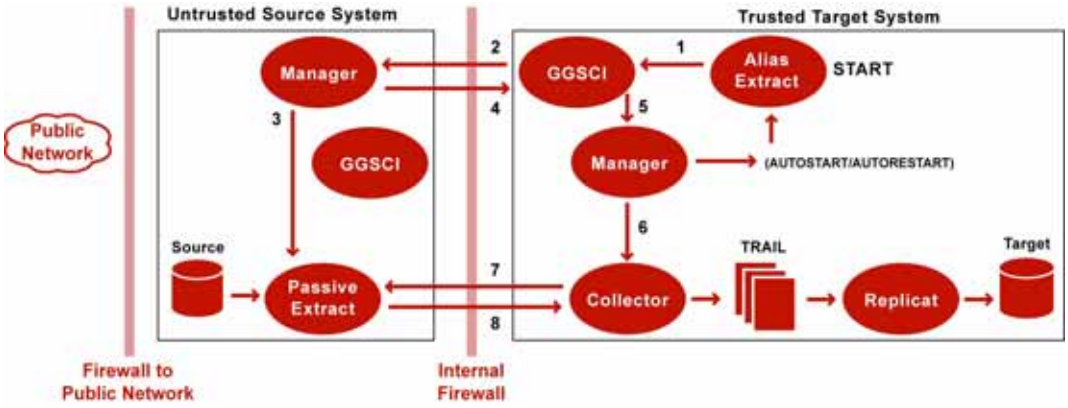
everyone but Oracle GoldenGate Administrators.

Using target system connection initiation

When a target system resides inside a trusted intranet zone, initiating connections from the source system (the standard Oracle GoldenGate method) may violate security policies if the source system is in a less trusted zone. It also may violate security policies if a system in a less trusted zone contains information about the ports or IP address of a system in the trusted zone, such as that normally found in an Oracle GoldenGate Extract parameter file.

In this kind of intranet configuration, you can use a *passive-alias Extract* configuration. Connections are initiated from the target system inside the trusted zone by an *alias Extract* group, which acts as an alias for a regular Extract group on the source system, known in this case as the *passive Extract*. Once a connection between the two systems is established, data is processed and transferred across the network by the passive Extract group in the usual way.

Figure 16 Connection initiation from trusted network zone



1. An Oracle GoldenGate user starts the alias Extract on the trusted system, or an AUTOSTART or AUTORESTART parameter causes it to start.
2. GGSCI on the trusted system sends a message to Manager on the less trusted system to start the associated passive Extract. The host name or IP address and port number of the Manager on the trusted system are sent to the less trusted system.
3. On the less trusted system, Manager starts the passive Extract, and the passive Extract finds an open port (according to rules in the DYNAMICPORTLIST Manager parameter) and listens on that port.
4. The Manager on the less trusted system returns that port to GGSCI on the trusted system.
5. GGSCI on the trusted system sends a request to the Manager on that system to start a Collector process on that system.
6. The target Manager starts the Collector process and passes it the port number where Extract is listening on the less trusted system.

7. Collector on the trusted system opens a connection to the passive Extract on the less trusted system.
8. Data is sent across the network from the passive Extract to the Collector on the target and is written to the trail in the usual manner for processing by Replicat.

Configuring the passive Extract group

The passive Extract group on the less trusted source system will be one of the following, depending on which one is responsible for sending data across the network:

- A solo Extract group that reads the transaction logs and also sends the data to the target, or:
- A data pump Extract group that reads a local trail supplied by a primary Extract and then sends the data to the target. In this case, there are no special configuration requirements for the primary Extract, just the data pump.

To create an Extract group in passive mode, use the standard ADD EXTRACT command and options, but add the PASSIVE keyword in any location relative to other command options. Examples:

```
ADD EXTRACT fin, TRANLOG, BEGIN NOW, PASSIVE, DESC "passive Extract"
ADD EXTRACT fin, PASSIVE, TRANLOG, BEGIN NOW, DESC "passive Extract"
```

To configure parameters for the passive Extract group, create a parameter file in the normal manner, except:

- *Exclude* the RMTHOST parameter, which normally would specify the host and port information for the target Manager.
- Use the optional RMTHOSTOPTIONS parameter to specify any compression and encryption rules. For information about the RMTHOSTOPTIONS options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

For more information about configuring an Extract group, see Chapter 14.

Configuring the alias Extract group

The alias Extract group on the trusted target does not perform any data processing activities. Its sole purpose is to initiate and terminate connections to the less trusted source. In this capacity, the alias Extract group does not use a parameter file nor does it write processing checkpoints. A checkpoint file is used only to determine whether the passive Extract group is running or not and to record information required for the remote connection.

To create an Extract group in alias mode, use the ADD EXTRACT command without any other options except the following:

```
ADD EXTRACT <group>
, RMTHOST {<host name> | <IP address>}
, MGRPORT <port>
[, RMTNAME <name>]
[, DESC "<description>"]
```

The RMTHOST specification identifies this group as an alias Extract, and the information is written to the checkpoint file. The <host name> and <IP address> options specify either the name or IP address of the source system. MGRPORT specifies the port on the source system

where Manager is running.

The alias Extract name can be the same as that of the passive Extract, or it can be different. If the names are different, use the optional RMTNAME specification to specify the name of the passive Extract. If RMTNAME is not used, Oracle GoldenGate expects the names to be identical and writes the name to the checkpoint file of the alias Extract for use when establishing the connection.

Error handling for TCP/IP connections is guided by the TCPERRS file on the target system. It is recommended that you set the response values for the errors in this file to RETRY. The default is ABEND. This file also provides options for setting the number of retries and the delay between attempts. For more information, see page 172.

Starting and stopping the passive and alias processes

To start or stop Oracle GoldenGate extraction in the passive-alias Extract configuration, start or stop the alias Extract group from GGSCI on the target.

```
START EXTRACT <alias group name>
```

Or...

```
STOP EXTRACT <alias group name>
```

The command is sent to the source system to start or stop the passive Extract group. Do not issue these commands directly against the passive Extract group. You can issue a KILL EXTRACT command directly for the passive Extract group.

When using the Manager parameters AUTOSTART and AUTORESTART to automatically start or restart processes, use them on the target system, not the source system. The alias Extract is started first and then the start command is sent to the passive Extract.

Managing extraction activities

Once extraction processing has been started, you can manage and monitor it in the usual manner by issuing commands against the passive Extract group from GGSCI on the source system. The standard GGSCI monitoring commands, such as INFO and VIEW REPORT, can be issued from either the source or target systems. If a monitoring command is issued for the alias Extract group, it is forwarded to the passive Extract group. The alias Extract group name is replaced in the command with the passive Extract group name. For example, INFO EXTRACT alias becomes INFO EXTRACT passive. The results of the command are displayed on the system where the command was issued.

Other considerations

When using a passive-alias Extract configuration, these rules apply:

- In this configuration, Extract can only write to one target system.
- This configuration can be used in an Oracle RAC installation by creating the Extract group in the normal manner (using the THREADS option to specify the number of redo threads).
- The ALTER EXTRACT command cannot be used for the alias Extract, because that group does not do data processing.

- To use the DELETE EXTRACT command for a passive or alias Extract group, issue the command from the local GGSCI.
- Remote tasks, specified with RMTTASK in the Extract parameter file and used for some initial load methods, are not supported in this configuration. A remote task requires the connection to be initiated from the source system and uses a direct connection between Extract and Replicat.

CHAPTER 11

Mapping and manipulating data

You can integrate data between source and target tables by:

- Mapping source objects to target objects
- Selecting records and columns
- Selecting and converting SQL operations
- Mapping dissimilar columns
- Using transaction history
- Testing and transforming data
- Using tokens

Limitations of support

The following are limitations to the support of data mapping and manipulation.

- When the size of a large object exceeds 4K, Oracle GoldenGate stores the data in segments within the Oracle GoldenGate trail. The first 4K is stored in the base segment, and the rest is stored in a series of 2K segments. Oracle GoldenGate does not support the filtering, column mapping, or manipulation of large objects of this size. Full Oracle GoldenGate functionality can be used for objects that are 4K or smaller.
- 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. TABLE is used in the Extract parameter file, and MAP is used in the Replicat parameter file.

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. For more information about how to create a source-definitions or target-definitions file, see Chapter 13 on page 174.

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

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 added overhead on the source system, you can configure mapping and conversion on the target system or on an intermediary system. However, in the case where there are multiple sources and one target, it might be preferable 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" on page 174.

Mapping and conversion on NonStop systems

If you are mapping or converting data from a Windows or UNIX system to a NonStop target, the mapping or conversion must be performed on the Windows or UNIX source system. Replicat for NonStop cannot convert two-part table names and data types to the three-part names that are used on the NonStop platform. Extract can format the trail data with NonStop names and target data types.

Globalization considerations when mapping data

When planning to map and convert data between databases, take into consideration what is supported or not supported by Oracle GoldenGate in terms of globalization. Understanding how character sets and locale are applied and converted will help you obtain accurate results.

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" on page 285 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.

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.

Character-set conversion for column data is limited to column-to-column mapping as specified in the COLMAP or USEDEFAULTS clauses of the Replicat MAP parameter. No mapping or conversion is performed by Extract or a data pump.

Replicat performs the conversion for all target database types except Oracle. Replicat allows Oracle to perform its own conversions in all cases except where the source is DB2 on z/OS and the trail is written in EBCDIC. In that case, Replicat performs the conversion, because Oracle does not convert from EBCDIC encoding.

> **NOTE** For an Oracle target to perform the conversion, the Replicat parameter file must contain a SETENV parameter that sets the NLS_LANG environment variable to the character set of the *source* database. For more information, see the Oracle GoldenGate *GoldenGate for Oracle Installation Guide*.

If the trail is written in an older format (typically by an older version of Extract), the character set for character-type columns might not be available to send to Replicat. In this case, Replicat does not perform conversion by default, except between EBCDIC and ASCII. It uses the character set of the host operating system for CHAR, VARCHAR2 and CLOB columns, and it uses UTF-16 big endian for NCHAR, NVARCHAR2 and NCLOB columns. To enable an accurate conversion, consult Oracle Support to use the internal parameter _TRAILCHARSET.

Preservation of locale

Oracle GoldenGate takes the locale of the database into account when comparing case-insensitive object names. See “Supported locales” on page 292 for a list of supported locales.

Using escape sequences for specific characters

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).

To use an escape sequence

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

- Must begin with a lowercase u, followed by exactly four 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)

Example \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)

Example \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)

Example \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.

Supporting case and special characters in column names

By default, Oracle GoldenGate treats any string within double quotes as a literal. To support column names that are case-sensitive or contain special characters, you can use the USEANSISQLQUOTES parameter. USEANSISQLQUOTES enables Oracle GoldenGate to follow SQL-92 rules for using quotation marks to delimit identifiers and literal strings. With USEANSISQLQUOTES enabled, Oracle GoldenGate treats a string within double quotes as a column name, and it treats a string within single quotes as a literal. This support applies globally to all processes in the Oracle GoldenGate instance. For more information about usage and limitations, see USEANSISQLQUOTES in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Using table-level column mapping

Use the COLMAP option of the MAP and TABLE parameters to:

- explicitly map source columns to target columns that have different names.
- specify default column mapping when an explicit column mapping is not needed.

COLMAP provides instructions for selecting, mapping, translating, and moving data from a source column into a target column. Within a COLMAP statement, you can employ any of the Oracle GoldenGate column-conversion functions to transform data for the mapped columns.

Specifying data definitions

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, i.e. 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.

See "Associating replicated data with metadata" on page 174.

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. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

COLMAP availability

The COLMAP option is available with the following parameters:

Extract	Replicat
TABLE	MAP

Syntax

```
TABLE <table spec>, TARGET <table spec>,
COLMAP ([USEDEFAULTS, ] <target column> = <source expression>);
```

Or...

```
MAP <table spec>, TARGET <table spec>,
COLMAP ([USEDEFAULTS, ] <target column> = <source expression>);
```

Table 14 TABLE and MAP arguments

Argument	Description
`TARGET <table spec>`	The target owner and table. Always required for MAP. Required for TABLE if COLMAP is used.
`<target column>`	The name of the target column to which you are mapping data. For information about supporting case-sensitivity or special characters in column names, see "Supporting case and special characters in column names" on page 146.
`<source expression>`	Can be any of the following to represent the data that is to be mapped: ◆ Numeric constant, such as 123 ◆ String constant enclosed within double quotes, such as "ABCD". Single quotes, such as 'ABCD', can be used if the USEANSISQLQUOTES parameter is used in the GLOBALS file. ◆ The name of a source column, such as ORD_DATE. To use double quotes for a column name that is case-sensitive or contains special characters, such as "Ord Date", use the USEANSISQLQUOTES parameter in the GLOBALS file. ◆ An expression using an Oracle GoldenGate column-conversion function, such as: @STREXT (COL1, 1, 3) For information about supporting case-sensitivity or special characters in column names, see "Supporting case and special characters in column names" on page 146.
`USEDEFAULTS`	Applies default mapping rules to map source and target columns automatically if they have the same name. USEDEFAULTS eliminates the need to map every target column explicitly, whether or not the source column has the same name. Transformation of data types is automatic based on the data-definitions file that was created with the defgen utility. Use an explicit map or USEDEFAULTS, but not both for the same set of columns. For more information about default column mapping, see "Using default column mapping" on page 151. For more information about TABLE and MAP, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Example 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 15 Sample column mapping

Parameter statement	Description
`MAP SALES.ACCTBL,` `TARGET SALES.ACCTTAB,`	*1.* Maps the source table ACCTBL to the target table ACCTTAB.
`COLMAP(`	*2.* Begins the COLMAP statement.
`USEDEFAULTS,`	*3.* Moves source columns as-is when the target column names are identical.
`NAME = CUST_NAME,`	*4.* Maps the source column CUST_NAME to the target column NAME.

`TRANSACTION_DATE =`
`@DATE("YYYY-MM-DD", "YY",`
`YEAR, "MM", MONTH, "DD", DAY),`
	5. 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))`
`;`
	6. 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.

Using global column mapping

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 Sybase, MySQL, SQL Server, PostgreSQL, 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 in quotes.
- For Oracle, DB2, and SQL/MX, where names can be either case-sensitive or case-insensitive in the same database, COLMATCH requires an exact case and name match when a name is in quotes.

The following clarifies the support for Oracle, DB2, and SQL/MX.

COLMATCH with NAMES

If the NAMES option is used, a quoted column name is treated as case-sensitive, and an unquoted column name is treated as case-insensitive. The following specifies a case-insensitive source column name abc, and a case-sensitive target column name "abc".

```
COLMATCH NAMES "abc" = abc
```

The following specifies a case-insensitive source column name abc, and a case-insensitive target column name xyz.

```
COLMATCH NAMES xyz = abc
```

COLMATCH with PREFIX or SUFFIX

You can specify a double-quoted or unquoted PREFIX and SUFFIX for COLMATCH to ignore. A quoted prefix or suffix is treated as case-sensitive, and an unquoted prefix or suffix is treated as case-insensitive. Prefix or suffix case-sensitivity only affects the prefix or suffix, but not the rest of the column specification.

The following example specifies a case-insensitive prefix p_ 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".

```
COLMATCH PREFIX p_
```

The following example specifies a case-sensitive suffix _k 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".

```
SUFFIX "_k"
```

Syntax

```
COLMATCH
{NAMES <target column> = <source column> |
PREFIX <prefix> |
SUFFIX <suffix> |
RESET}
```

Table 16 COLMATCH options

Argument	Description
`NAMES <target column> = <source column>`	Maps based on column names.
`PREFIX <prefix>`	Ignores the specified name prefix.
`SUFFIX <suffix>`	Ignores the specified name suffix.
`RESET`	Turns off previously defined COLMATCH rules for subsequent TABLE or MAP statements.

Example 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 17 COLMATCH example tables

Source Database		Target Database	
ACCT Table	**ORD Table**	**ACCOUNT Table**	**ORDER Table**
CUST_CODE	CUST_CODE	CUSTOMER_CODE	CUSTOMER_CODE
CUST_NAME	CUST_NAME	CUSTOMER_NAME	CUSTOMER_NAME
CUST_ADDR	ORDER_ID	CUSTOMER_ADDRESS	ORDER_ID
PHONE	ORDER_AMT	PHONE	ORDER_AMT
S_REP	S_REP	REP	REP
S_REPCODE	S_REPCODE	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 would be:

```
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 "Using default column mapping".
- 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.

Using 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 18.

Table 18 Defaults for target columns that cannot be matched

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

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.

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.

By default, literals must be enclosed within double quotes. For information about using SQL-92 rules to delimit column identifiers and literals, see “Supporting case and special characters in column names” on page 146.

Datetime columns

Datetime (DATE, TIME, and TIMESTAMP) columns can accept datetime and character columns,

as well as string literals. 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.FFFFFF.

By default, literals must be enclosed within double quotes. For information about using SQL-92 rules to delimit column identifiers and literals, see "Supporting case and special characters in column names" on page 146.

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 rows

To filter out or select rows for extraction or replication, use the FILTER and WHERE clauses of the following parameters.

Extract	Replicat
TABLE	MAP

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

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.

Syntax

```
TABLE <table spec>,
, FILTER (
[, ON INSERT | ON UPDATE| ON DELETE]
[, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]
, <filter clause>);
```

Or...

Syntax

```
MAP <table spec>, TARGET <table spec>,
, FILTER (
[, ON INSERT | ON UPDATE| ON DELETE]
[, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]
[, RAISEERROR <error_num>]
, <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. For more information about Oracle GoldenGate conversion functions, see "Testing and transforming data" on page 160.

- 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.

By default, literal strings must be enclosed in double quotes in FILTER. To use double-quotes for column names that are case-sensitive or contain special characters, and to use single quotes for literal strings, use the USEANSISQLQUOTES parameter in the GLOBALS file. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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.

Example 1 The following 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 2 The following uses the @STREQ function to extract records where a string is equal to 'JOE'. This example assumes that the USEANSISQLQUOTES parameter is used in the GLOBALS parameter file to apply SQL-92 rules for single and double quote marks.

```
TABLE ACCT.TCUSTORD, FILTER (@STREQ ("Name", 'joe') > 0);
```

Example 3 The following selects records in which the amount column is greater than 50 and executes the filter on updates and deletes.

```
TABLE ACT.TCUSTORD, FILTER (ON UPDATE, ON DELETE, AMOUNT > 50);
```

Example 4 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) based on the ID column of the source acct table. Note that object names are case-sensitive in this case.

(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));
```

Example 5 You can combine several FILTER clauses in one MAP or TABLE statement, as shown in Table 19, 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.

Table 19 Using multiple FILTER statements

Parameter file	Description
`REPERROR (9999, EXCEPTION)`	*1.* Raises an exception for the specified error.
`MAP OWNER.SRCTAB,` `TARGET OWNER.TARGTAB,`	*2.* Starts the MAP statement.

`SQLEXEC (ID CHECK, ON UPDATE,`
`QUERY " SELECT COUNT FROM TARGTAB "`
`"WHERE PKCOL = :P1 ",`
`PARAMS (P1 = PKCOL)),`
	3. Performs a query to retrieve the present value of the count column whenever an update is encountered.

`FILTER (BALANCE > 15000),`	*4.* Uses a FILTER clause to select rows where the balance is greater than 15000.
`FILTER (ON UPDATE, BEFORE.COUNT =` `CHECK.COUNT)`	*5.* Uses another FILTER clause to ensure that the value of the source count column before an update matches the value in the target column before applying the target update.
`;`	*6.* The semicolon concludes the MAP statement.

`MAP OWNER.SRCTAB,`
`TARGET OWNER.TARGEXC,`
`EXCEPTIONSONLY,`
`COLMAP (USEDEFAULTS,`
`ERRTYPE = "UPDATE FILTER FAILED");`
	7. Designates an exceptions MAP statement. The REPERROR clause for error 9999 ensures that the exceptions map to targexc will be executed.

Selecting rows with a WHERE clause

Use any of the elements in Table 20 in a WHERE clause to select or exclude rows (or both) based on a conditional statement. Each WHERE clause must be enclosed within parentheses.

Table 20 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" on page 156.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses ( ) for logical grouping of multiple elements.

By default, literal strings must be enclosed in double quotes in WHERE. To use double-quotes for column names that are case-sensitive or contain special characters, and to use single quotes for literal strings, use the USEANSISQLQUOTES parameter in the GLOBALS file. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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" on page 239).

Syntax

```
TABLE <table spec>, WHERE (<WHERE clause>);
```

Or...

```
MAP <table spec>, TARGET <table spec>, WHERE (<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; thus, the letter case used in the parameters does not matter.

Ensuring data availability for filters

If the database uses *compressed updates* (where only values for *changed* columns appear

in the transaction log), there can be errors when the missing 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 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 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; thus, the table names are in quote marks. To use quote marks for case-sensitive column names, the USEANSISQLQUOTES parameter must be used.

To reference the before value

1. Use the BEFORE keyword, then a dot (.), then 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 uses compressed updates, using the BEFORE prefix results in a "column missing" condition and the column map is executed as if the column were not in the record. To ensure that column values are available, see "Ensuring data availability for filters" on page 156.

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 "Dave." The last operation deletes the record, so there is no way to find out Dave's account history or his ending balance.

Figure 17 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.

Example Using the sample series of transactions shown in Figure 17 on page 159, 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	GETUPDATEBEFORES TABLE ACCOUNT.CUSTOMER;
Replicat	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);

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 the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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. For more information about user exits, see page 239.

Oracle GoldenGate conversion functions take the following general syntax:

Syntax `@<function name> (<argument>)`

Syntax element	Description
`@<function name>`	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.
`<argument>`	Function arguments can contain the following:

Argument element	Example
A numeric constant	`123`
A string literal	`"ABCD" or 'ABCD'` (depending on whether USEANSISQLQUOTES is used in GLOBALS)
The name of a source column	`PHONE_NO or "Phone_No"` (depending on whether USEANSISQLQUOTES is used in GLOBALS)
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

By default, literal strings must be enclosed in double quotes in a column-conversion function. To use double-quotes for column names that are case-sensitive or contain special characters, and to use single quotes for literal strings, use the USEANSISQLQUOTES parameter in the GLOBALS file. This parameter applies SQL-92 rules for literals and identifiers. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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.

Example This statement would fail because it uses @IF, 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" on page 163.

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.

Example This example computes the time that an order is filled.

```
ORDER_FILLED = @DATE (
    "YYYY-MM-DD:HH:MI:SS",
    "JTS",
    @DATE ("JTS",
    "YYMMDDHHMISS",
    ORDER_TAKEN_TIME) +
    ORDER_MINUTES * 60 * 1000000)
```

NOTE By default, the hard-coded dates shown in this example are treated as literals because they are within double quotes. If using the USEANSISQLQUOTES parameter, the literal dates must be enclosed within single quotes.

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

The @COMPUTE keyword is not required when an expression is passed as a function argument.

Example

```
@STRNUM ((AMOUNT1 + AMOUNT2), LEFT)
```

The following expression would return 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 21 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

Table 21 Conversion functions for numbers and characters (continued)

Purpose	Conversion Function
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.

Example 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

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.

Example 1 The following example returns a NULL into target column ITEM.

```
ITEM = @COLSTAT (NULL)
```

Example 2 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.

Example

```
@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.

Example
```
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

Use @CASE to select a value depending on a series of value tests.

Example
```
@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

> **NOTE** By default, the strings shown in this example are treated as literals because they are within double quotes. If using the USEANSISQLQUOTES parameter, the strings must be enclosed within single quotes.

Using @VALONEOF

Use @VALONEOF to compare a column or string to a list of values.

Example
```
@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.

Example
```
@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 extract 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

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 2,000 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. 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 double quotes (or within single quotes if USEANSISQLQUOTES is in use) or the result of an Oracle GoldenGate column-conversion function (see the following example). 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.

Example

```
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. Note that the arguments are enclosed within double quotes; however, they must be in single quotes if the USEANSISQLQUOTES parameter is in use. For more information about USEANSISQLQUOTES and @GETENV, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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>"))
```

Example 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 are enclosed within double quotes; however, they must be in single quotes if the USEANSISQLQUOTES parameter is in use. For more information about USEANSISQLQUOTES, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

```
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 would look similar to the following within the record header in the trail:

```
User tokens:
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
tk-pos         :3604496
tk-rba         :4058
tk-table       :oratest
tk-optype      :insert
```

CHAPTER 12

Handling Oracle GoldenGate processing errors

This chapter contains information about how to handle processing errors. Oracle GoldenGate reports processing errors in several ways by means of its monitoring and reporting tools. For more information about these tools, see Chapter 17 on page 247.

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. For a complete list, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

- FETCHOPTIONS
- WARNLONGTRANS
- DBOPTIONS
- TRANLOGOPTIONS

To handle extraction errors that relate to DDL operations, use the DDLERROR parameter. For more information, see page 178.

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".
- 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. Replicat processes other operations in the transaction that do not generate errors. The exceptions are TRANSDISCARD and TRANSEXCEPTION: These options affect all records in a transaction in which any record generates an error. (The ABEND option also applies to the entire transaction, but does not apply error handling.)

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 the following options of the MAP parameter:

- MAP with EXCEPTIONSONLY
- MAP with MAPEXCEPTION

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.

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. In addition, 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.

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, use 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.

Example 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.

There are two MAP statements:

- 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 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 will be captured to the fin.trxexceptions table using the designated mapping.

```
MAP src.trx*, TARGET trg.*,
MAPEXCEPTION (TARGET fin.trxexceptions,
COLMAP (USEDEFAULTS,
ACCT_NO = ACCT_NO,
OPTYPE = @GETENV ("LASTERR", "OPTYPE"),
DBERR = @GETENV ("LASTERR", "DBERRNUM"),
DBERRMSG = @GETENV ("LASTERR", "DBERRMSG")
)
);
```

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 "Handling DDL processing errors" on page 178.

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

Figure 18 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 22:

Table 22 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 RETRY or ABEND.
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.

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

For help with resolving Oracle GoldenGate errors, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide* Guide.

CHAPTER 13

Associating replicated data with metadata

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.

Configuring Oracle GoldenGate to assume identical metadata

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 owners and/or names can be different.

Rules for tables to be considered identical

For source and target structures to be identical, they must:

- be of the same database type, i.e. 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 the Oracle GoldenGate *Windows and UNIX Reference Guide.*

```
-- Specifies the group name.
REPLICAT acctrep
-- Specifies database login with encrypted password.
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
-- 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;
```

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. 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

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, mapping and conversion can be performed by Extract, a data-pump Extract, or Replicat, but is usually performed by Replicat on the target system.
- When replicating to a NonStop Server target from any Windows, UNIX, or Linux-based database system, mapping and conversion must be performed on the Windows, UNIX, or Linux system: Only Extract can convert two-part table names and data types to the three-part names that are used on the NonStop platform.

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 or transformation on an intermediary system, use a *source-definitions file* and a *target-definitions file* on that system to supply source and target definitions to the Extract data pump.
  - 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.

See the topology-specific configuration instructions in this documentation for more information about using a definitions file in your intended replication topology.

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.

Evaluating character sets for the definitions file

By default, DEFGEN writes the definitions file in the character set of the local operating system. If the remote system to which you are transfering the file has the same or equivalent character set as the local system, the definitions file can probably be created on one system and moved to the remote system without any encoding-related problems. 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.

When the character set of one operating system is a superset of the other one, you can do the following:

- If the target character set is a superset of the source character set, and you need to use a definitions file on the source system, create the definitions file on the target system and use it on the source as-is (without conversion).
- If the source character set is a super set of the target character set, and you need to use a definitions file on the target (or for an intermediary data pump), create the definitions file on the source system and use it on the target as-is (without conversion).

You do not need to create or update those definitions files to the character set of the other operating system because the original character set is a superset and this method prevents incompatibility errors.

Alternatively, consider the case of UNIX ISO-8859-1 Latin1 and Windows code page 1252 Latin1. These character sets are equivalent and, in most cases, compatible. However, if the file contains the Euro symbol (€: U+20AC), for example, there is an incompatibility because the Euro symbol is available on Windows 1252 as 0x80 but not on UNIX ISO-8859-1. Many other character sets have little or no compatibility between them. To overcome these incompatibilities, DEFGEN enables you to change the character set of the definitions file.

Changing the character set of a new definitions file

To generate the definitions file in a character set other than the default one of the operating system, specify the desired character set (typically that of the remote operating system) with the CHARSET option of the DEFSFILE parameter. You specify this parameter when you create the DEFGEN parameter file. The definitions file will be written in the specified character set.

Changing the character set of existing definitions files

In the case of an existing definitions file that is transfered 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.

Configuring Oracle GoldenGate to capture data-definitions

To configure Oracle GoldenGate to use a data-definitions file and template (if needed), you will:

- Configure DEFGEN
- Run DEFGEN
- Transfer the definitions file to the remote system
- Specify the definitions file

NOTE Do not create a data-definitions file for Oracle sequences. It is not needed and DEFGEN does not support it.

Configure DEFGEN

Perform these steps on the system from which you want to obtain metadata definitions.

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 23 in the order shown, starting a new line for each parameter statement.

Table 23 DEFGEN parameters

Parameter	Description
`CHARSET <character set>`	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.
`DEFSFILE <full_pathname> [APPEND | PURGE] [CHARSET <character set>]` ◆ 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.	Specifies the relative or fully qualified name of the data-definitions file that is to be the output of DEFGEN. See DEFSFILE in the Oracle GoldenGate *Windows and UNIX Reference Guide* for important information about these parameter options and their effect on character sets. For information about character sets and Oracle GoldenGate in general, see "Evaluating character sets for the definitions file" on page 176.
`[{SOURCEDB | TARGETDB} <dsn>] USERID <user>[, PASSWORD <password> [<encryption options>]]` ◆ SOURCEDB | TARGETDB specifies a data source name, if required as part of the connection information. Not required for Oracle. ◆ USERID and optional PASSWORD provide database authentication, as needed.	Specifies database connection information. For more information about SOURCEDB and USERID options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Table 23 DEFGEN parameters (continued)

Parameter	Description
`TABLE <owner>.<table> [, {DEF | TARGETDEF} <template name>];` Where: ◆ <owner> is the schema name. ◆ <table> is the name of a table or a group of tables defined with an asterisk wildcard, for which definitions will be output to the definitions file. DEFGEN does not support the question mark wildcard. ◆ [, {DEF | TARGETDEF} <template name>] additionally creates a definitions template based on the metadata of this table. This option is not supported for initial loads.	Specifies 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 is preserved for case-sensitive databases. For instructions on wildcarding and case-sensitivity, see "Specifying object names in Oracle GoldenGate input" on page 35. Specify a source table(s) if generating a source-definitions file or a target table(s) if generating a target-definitions file. To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter. **Note:** DEFGEN does not support UDTs.

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 dirprm/defgen.prm is the relative or full path name of the DEFGEN parameter file.
   - reportfile dirrpt/defgen.rpt sends output to the screen and to the designated report file. You can omit this argument just to print to screen.
   - 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.

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.

For the correct way to specify multiple definitions files, see "Examples of using a definitions file" on page 180.

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

Example 1 Create source-definitions files for use on target

The following example 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
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
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
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
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.

Example 2 Create target-definitions files for use on source

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
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
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
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
RMTHOST sysb, MGRPORT 7890, ENCRYPT AES192 KEYNAME mykey1
ENCRYPTTRAIL AES192 KEYNAME mykey2
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;
```

Example 3 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 acc.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).

Example 4 Create multiple source-definitions files for use on target

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.

```
DEFSFILE C:\ggs\dirdef\sales.def
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
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.

```
DEFSFILE C:\ggs\dirdef\admin.def
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
TABLE hr.*;
```

The tables for the first and second definitions file are mapped in the same Replicat parameter file as follows:

```
REPLICAT acctrep
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
SOURCEDEFS c:\ggs\dirdef\sales.def
MAP ord.*, TARGET ord.*;
SOURCEDEFS c:\ggs\dirdef\admin.def
MAP hr.*, TARGET hr.*;
```

Configuring Oracle GoldenGate to use a combination of similar and dissimilar definitions

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.

Example 5 This 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
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
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;
```

CHAPTER 14

Configuring online change synchronization

Overview of online change synchronization

Online change synchronization extracts and replicates data changes continuously to maintain a near real-time target database. The following summarizes the steps required to configure online change synchronization:

- Create a checkpoint table
- Create one or more Extract groups.
- Create an Extract parameter file.
- Create a trail.
- Create a Replicat group.
- Create a Replicat parameter file.

Initial synchronization

After you configure your change-synchronization groups and trails following the directions in this chapter, see "Running an initial data load" on page 200 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.

Configuring process groups for best performance

Develop business rules that specify the acceptable amount of lag between when changes are made within your source applications and when those changes are applied to the target database. These rules will determine the number of parallel Extract and Replicat processes that are required to enable Oracle GoldenGate to perform at its best.

Gather the size and activity rates for all of the tables that you intend to replicate with Oracle GoldenGate.

- Assign one Extract group to all of the tables that have low activity rates.
- Assign a dedicated Extract group to each table that has high activity rates.

Configure these Extract groups to work with dedicated data pumps and Replicat groups. For more information about configuring Oracle GoldenGate for best performance, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

Naming conventions for groups

When naming Oracle GoldenGate process groups, follow these rules:

- You can use up to eight ASCII characters, including non-alphanumeric characters such as the underscore (_). Any ASCII character can be used, so long as the operating system allows that character to be in a filename. This is because a group is identified by its associated checkpoint file.
- The following ASCII characters are not allowed in a file name:

  ```
  { \ / : * ? " < > | }
  ```
- 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, group 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 group name (and its parameter file name if explicitly defined in the ADD command) must be all uppercase or all lowercase. Mixed case group names and parameter file names will result in errors when starting the process.
- Use only one word.
- Do not use the word “port” as a group name. However, you can use the string “port” as part of the group name.
- You can include a number in a group name. However, be aware 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. You can place a numeric value at the beginning of a group name, such as 1_fin, 1fin, and so forth without concern.

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.

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 table(s) 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".
- You can provide specific checkpoint table instructions when you create any given Replicat group:
  - To use a specific checkpoint table for a group, use the CHECKPOINTTABLE argument in the ADD REPLICAT command. A checkpoint table specified in ADD REPLICAT 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 in the ADD REPLICAT command. If you opt not to use a checkpoint table, the checkpoints will be maintained in a checkpoint file on disk, but you introduce the risk of data inconsistency. For more information, see"Overview of checkpoints" on page 14.

Regardless of how you want to implement the checkpoint table, you must create it in the target database prior to using the ADD REPLICAT command. See "To add a checkpoint table to the target database".

To specify a default checkpoint table in the GLOBALS file

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 (case does not matter):

   ```
   CHECKPOINTTABLE <owner>.<table>
   ```

   Where: <owner>.<table> is the owner and name for the default checkpoint table. The name can be anything supported by the database.

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.

To add a checkpoint table to the target database

NOTE 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 following command to log into the database.

   ```
   DBLOGIN [SOURCEDB <dsn>][, USERID <db_user>][, PASSWORD <pw>[<encryption
   options]]
   ```

Where:

- SOURCEDB <dsn> supplies a data source name, if required as part of the connection information.
- USERID <db_user>, PASSWORD <pw>, and <encryption options> supply database credentials and encryption information, if required. PASSWORD is not required for NonStop SQL/MX or DB2.

This user must have CREATE TABLE permissions.

2. In GGSCI, issue the following command to add the checkpoint table to the database.

```
ADD CHECKPOINTTABLE [<owner>.<table>]
```

Where:

<owner>.<table> is the owner and name of the table. 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.

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.

To create a regular, passive, or data pump Extract group

```
ADD EXTRACT <group name>
{, <datasource>}
{, BEGIN <start point>} | {<position point>}
[, PASSIVE]
[, THREADS <n>]
[, PARAMS <pathname>]
[, REPORT <pathname>]
[, DESC "<description>"]
```

Where:

- <group name> is the name of the Extract group. A group name is required, can contain up to eight characters, and is not case-sensitive. See page 185 for more information.
- <datasource> is required to specify the source of the data to be extracted. Use one of the following:
  - `TRANLOG [<bsds name>]` specifies the transaction log as the data source. Use for all databases except Teradata. Use the <bsds> option for DB2 running on z/OS to specify the Bootstrap Data Set file name of the transaction log. When using this option for Oracle Enterprise Edition version 10.2 and later, 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).

- INTEGRATED TRANLOG specifies that this Extract will operate in integrated capture mode to receive logical change records (LCR) from a database logmining server. This parameter applies only to Oracle databases. For information on using integrated capture, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.
- VAM specifies that the Extract API known as the *Vendor Access Module* (VAM) will interface with the Teradata Access Module (TAM). Use for Teradata databases.
- VAMTRAILSOURCE <VAM trail name> to specify a VAM trail. Use for Teradata extraction in maximum protection mode to create a VAM-sort Extract group. For more information, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- 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. 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[.CCCCCC]]> 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.

    NOTE Do not use the BEGIN parameter for a Teradata source.

- <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, consult the ADD EXTRACT documentation in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- PASSIVE indicates that the group is a passive Extract. When using PASSIVE, you must also use an alias Extract. See page 138 for more information. This option can appear in any order among other ADD EXTRACT options.
- THREADS <n> is required only for 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.

To create an alias Extract group

```
ADD EXTRACT <group name>
, RMTHOST {<host name> | <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 1 Log-based extraction

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 2 Teradata extraction, primary Extract

This example creates an Extract group named "finance" that performs in either Teradata maximum performance or Teradata maximum protection mode. No BEGIN point is used for Teradata sources.

```
ADD EXTRACT finance, VAM
```

Example 3 Teradata extraction, VAM-sort Extract

This example creates a VAM-sort Extract group named "finance." The process reads from VAM trail /ggs/dirdat/vt.

```
ADD EXTRACT finance, VAMTRAILSOURCE /ggs/dirdat/vt
```

Example 4 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, EXTTRAILSOURCE c:\ggs\dirdat\lt
```

Example 5 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 6 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\dirdat\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\dirdat\lt, PASSIVE
```

Example 7 Alias Extract group

This example creates an alias Extract group named "alias."

```
ADD EXTRACT alias, RMTHOST sysA, MGRPORT 7800, RMTNAME finance
```

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:

- VAM-sort Extract: Extracts from a local trail that is created as a VAM trail (for Teradata source databases). For more information, see the Oracle GoldenGate *Teradata Installation and Setup Guide*
- 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. You link tables specified with a TABLE statement to a trail specified with an EXTTRAIL or RMTTRAIL parameter statement in the Extract parameter file. For more information about Oracle GoldenGate trails, see page 13.

To define a trail

In GGSCI on the source system, issue the following command.

```
ADD {RMTTRAIL | EXTTRAIL} <pathname>, EXTRACT <group name>
[, 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 or a VAM trail that is linked to a primary Extract which interacts with a Teradata Access Module (TAM). For more information about the Teradata configuration, see the Oracle GoldenGate *Teradata Installation and Setup Guide*
- <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\dirdat\rt. Oracle GoldenGate appends a serial number to each trail file as it is created during processing. Typically, trails are stored in the dirdat sub-directory of the Oracle GoldenGate directory.
- EXTRACT <group name> 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 This example creates a VAM trail named /ggs/dirdat/vt for Extract group "extvam."

```
ADD EXTTRAIL /ggs/dirdat/vt, EXTRACT extvam
```

Example This example creates a local trail named /ggs/dirdat/lt for Extract group "ext."

```
ADD EXTTRAIL /ggs/dirdat/lt, EXTRACT ext
```

Example This example creates a trail named c:\ggs\dirdat\rt for Extract group "finance," with each file sized at approximately 50 megabytes.

```
ADD RMTTRAIL c:\ggs\dirdat\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. For more information, see page 138.

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 Table 24 in the order shown, starting a new line for each parameter statement. Some parameters apply only for certain configurations.

Table 24 Online change-extraction parameters

Parameter	Description
`EXTRACT <group name>` ◆ <group name> is the name of the Extract group that you created with the ADD EXTRACT command.	Configures Extract as an online process with checkpoints.

`[SOURCEDB <dsn>,]`
`[USERID <user id>][, PASSWORD <pw>[<encryption options>]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption options, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
PASSWORD is not required for NonStop SQL/MX or DB2.
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. For more information about passwords and encryption options, see view parameters in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
	This parameter 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.

Table 24 Online change-extraction parameters (continued)

Parameter	Description
`RMTHOST <hostname>, MGRPORT <portnumber>, [, ENCRYPT <algorithm> KEYNAME <keyname>]`	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). Not valid for a primary Extract group that interfaces with a Teradata Access Module and writes to a VAM trail. For more information, see the Oracle GoldenGate *Teradata Installation and Setup Guide* Not valid for a passive Extract group.
`ENCRYPTTRAIL <encryption options>`	Encrypts all trails that are specified after this entry. For encryption options, see the *Windows and UNIX Reference Guide*.
`RMTTRAIL <full_pathname> | EXTTRAIL <full_pathname>` ◆ 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).	Specifies a trail. If specifying multiple trails, follow each designation with the appropriate TABLE statements. EXTTRAIL is not valid for a passive Extract group. If trails or files will be of different versions, use the FORMAT option of RMTTRAIL or EXTTRAIL. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*. 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).

Table 24 Online change-extraction parameters (continued)

Parameter	Description
`DSOPTIONS { COMMITTEDTRANLOG, RESTARTAPPEND | CREATETRANLOG | SORTTRANLOG }`	Valid only for Teradata extraction. ◆ Use COMMITTEDTRANLOG, RESTART APPEND to indicate that Extract is receiving fully committed data in Teradata maximum performance mode. RESTARTAPPEND appends data to the end of the Oracle GoldenGate trail rather than rewriting data from a previous run. ◆ Use CREATETRANLOG to direct Extract to create and write to a local VAM trail in Teradata maximum protection mode. Use for a primary Extract group that interfaces with the Teradata Access Module. ◆ Use SORTTRANLOG to cause Extract to read from a local VAM trail and sort the data in commit order in maximum protection mode. Use only for a VAM-sort Extract group. For more information about the Teradata configuration, see the Oracle GoldenGate *Teradata Installation and Setup Guide*
`VAM <library name>, PARAMS ("<param>" [, "<param>"] [, ...])`	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")`
`PASSTHRU | NOPASSTHU`	(For a data pump) Specifies whether or not subsequent TABLE specifications will use normal or pass-through processing.
`SEQUENCE <owner>.<sequence>;` ◆ <owner> is the schema name. ◆ <sequence> is the name of the sequence.	Specifies an Oracle sequence to capture.

`TABLE <owner>.<table>;`
◆ <owner> is the schema name.
◆ <table> is the name of the table or a group of tables defined with wildcards.
To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.
	Specifies a table or tables for which to extract data changes.
	Schema names cannot be wildcarded. To extract data from tables in multiple schemas, use a separate TABLE statement for each schema. For example:
	`TABLE fin.*;`
	`TABLE hr.*;`

3. Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
4. Save and close the parameter file.

Creating an online 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 name>, EXTTRAIL <pathname>
[, BEGIN <start point> | , EXTSEQNO <seqno>, EXTRBA <rba>]
[, CHECKPOINTTABLE <owner.table>]
[, NODBCHECKPOINT]
[, PARAMS <pathname>]
[, REPORT <pathname>]
```

Where:

- <group name> is the name of the Replicat group. A group name is required, can contain up to eight characters, and is not case-sensitive. See page 185 for more information.
- EXTTRAIL <pathname> is the relative or fully qualified name of the trail that you defined with the ADD RMTTRAIL command.
- 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[.CCCCCC]]> as the format for specifying an exact timestamp as the begin point.
- EXTSEQNO <seqno>, EXTRBA <relative byte address> 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\aa000026, you would specify EXTSEQNO 26. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.
- 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 "Initial synchronization" on page 184).
- NODBCHECKPOINT specifies that this Replicat group will not use a checkpoint table.
- 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.

Example The following 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 25 in the order shown, starting a new line for each parameter statement.

Table 25 Online change-replication parameters

Parameter	Description
`REPLICAT <group name>` ◆ <group name> is the name of the Replicat group that you created with the ADD REPLICAT command.	Configures Replicat as an online process with checkpoints.
`{SOURCEDEFS <full_pathname>} | ASSUMETARGETDEFS` ◆ Use SOURCEDEFS if the source and target tables have different definitions. Specify the source data-definitions file generated by DEFGEN. See Chapter 13 for more information. ◆ Use ASSUMETARGETDEFS if the source and target tables have the same definitions.	Specifies how to interpret data definitions. For Oracle databases that use multi-byte character sets, you must use SOURCEDEFS (with a DEFGEN-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 page 150 for more information.
`DISCARDFILE <full_pathname> [, MEGABYTES <n>] [, PURGE]` ◆ <full pathname> is the relative or fully qualified name of the discard file. The default location is the dirrpt sub-directory of the Oracle GoldenGate directory. ◆ MEGABYTES <n> specifies the maximum size of the discard file. ◆ PURGE overwrites any existing discard files.	Specifies a file to which Replicat writes rejected record data, for example records that generated database errors. A discard file is optional but recommended.

Table 25 Online change-replication parameters (continued)

Parameter	Description
`[DEFERAPPLYINTERVAL <n><unit>]` ◆ <n> is a numeric value for the amount of time to delay. Minimum is set by the EOFDELAY parameter. Maximum is seven days. ◆ <unit> can be: S | SEC | SECS | SECOND | SECONDS | MIN | MINS | MINUTE | MINUTES | HOUR | HOURS | DAY | DAYS	Optional. Specifies an amount of time for Replicat to wait before applying captured transactions to the target system.

`[TARGETDB <dsn>,]`
`[USERID <user id>][, PASSWORD <pw>[<encryption options]]`
◆ TARGETDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption information, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. For more information about passwords and encryption options, see view parameters in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`DECRYPTTRAIL <encryption options>`	Decrypts the input trail that this Replicat reads. The <encryption options> must match those of the ENCRYPTTRAIL statement for this input trail.

`MAP <owner>.<table>,`
`TARGET <owner>.<table>[, DEF <template name>];`
◆ <owner> is the schema name.
◆ <table> is the name of a table or a wildcard definition for multiple tables.
◆ [, DEF <template name>] specifies a definitions template. (See Chapter 13.)
	Specifies a relationship between a source and target table or tables.
	Schema names cannot be wildcarded. To extract data from tables in multiple schemas, use a separate MAP statement for each schema. For example:
	`MAP fin.*, TARGET fin.*;`
	`MAP hr.*, TARGET hr.*;`
	To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.

3. [Enter any appropriate optional Replicat parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
4. Save and close the file.

Controlling online processes

To start and stop online processes, use GGSCI.

NOTE On Windows Server 2008 with User Account Control enabled, you will receive a UAC prompt when starting an Oracle GoldenGate process if Manager was not installed as a Windows service.

To start online processes for the first time

Typically, the first time that Oracle GoldenGate processes are started in a production setting is during the initial synchronization process, assuming source user applications must remain active. While the target is loaded with the source data, Oracle GoldenGate captures ongoing user changes and then reconciles them with the results of the load. For more information, see Chapter 15 on page 200.

> **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.

To start an online process

```
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*).

> **NOTE** When Extract is in PASSIVE mode, it can be started only by starting the associated alias Extract. See page 138 for more information.

See the Oracle GoldenGate *Windows and UNIX Reference Guide* for additional START REPLICAT options that can be used as needed to skip the first transaction in the trail or start at a specific transaction.

To auto-start a process

- Use AUTOSTART in the Manager parameter file to start one or more processes when Manager starts.
- Use AUTORESTART in the Manager parameter file to restart a process after a failure.

Both of these parameters reduce the need to start a process manually with the START command.

To stop an online process 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*).

> **NOTE** When an Extract is running in PASSIVE mode, it can only be stopped by stopping the associated alias Extract. See page 138 for more information.

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 a process group

After stopping an online process, you can delete the group. Deleting a group 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.

To delete an Extract group

1. Run GGSCI.
2. (Oracle Enterprise Edition 10.2 or later) Issue the DBLOGIN command as the Extract database user (or a user with the same privileges).

   ```
   DBLOGIN USERID <Extract_user>, PASSWORD <password>[<encryption options>]
   ```

3. Issue the following command.

   ```
   DELETE EXTRACT <group> [!]
   ```

The ! argument deletes all Extract groups that satisfy a wildcard without prompting.

To delete a Replicat group

1. If using a checkpoint table for this group, issue the following command from GGSCI to log into the database.

   ```
   DBLOGIN [SOURCEDB <dsn>] USERID <user>[, PASSWORD <password>[ <encryption
   options>]]
   ```

   Where:

   - SOURCEDB <dsn> supplies the data source name, if required as part of the connection information.
   - USERID <user>, PASSWORD <password>, and <encryption options> supply database credentials and encryption information, if required.

2. Issue the following command to delete the group.

   ```
   DELETE REPLICAT <group>
   ```

Instead of logging into the database with DBLOGIN, you can use the ! option with DELETE REPLICAT.

```
DELETE REPLICAT <group> !
```

DELETE REPLICAT deletes the checkpoint file, but preserves the checkpoints in the checkpoint table. The basic DELETE REPLICAT command commits the Replicat transaction, but the ! option prevents the commit.

CHAPTER 15

Running an initial data load

Overview of initial data load methods

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.

The initial load can be performed from an active source database. Users and applications can access and update data while the load is running. You can perform initial load from a quiesced source database if you delay access to the source tables until the target load is completed.

Supported load methods

You can use Oracle GoldenGate to load data in any of the following ways:

- "Loading data with a database utility" on page 203. The utility performs the initial load.
- "Loading data from file to Replicat" on page 204. Extract writes records to an extract file and Replicat applies them to the target tables. This is the slowest initial-load method.
- "Loading data from file to database utility" on page 209. Extract writes records to extract files in external ASCII format. The files are used as data files for input into target tables by a bulk load utility. Replicat creates the run and control files.
- "Loading data with an Oracle GoldenGate direct load" on page 214. Extract communicates with Replicat directly across TCP/IP without using a Collector process or files. Replicat applies the data through the database engine.
- "Loading data with a direct bulk load to SQL*Loader" on page 219. Extract extracts records in external ASCII format and delivers them directly to Replicat, which delivers them to Oracle's SQL*Loader bulk-load utility. This is the fastest method of loading Oracle data with Oracle GoldenGate.
- "Loading data with Teradata load utilities" on page 224. This is the preferred method for synchronizing two Teradata databases. The recommended utility is MultiLoad.

Using parallel processing in 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

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.

Prerequisites for initial load

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. See page 150 for more information about DDL support.

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:** To use the HANDLECOLLISIONS function to reconcile incremental data changes with 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. For more information, see "Configuring Manager and Network Communications" on page 18.

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. For more information, see Chapter 13.

Create change-synchronization groups

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.

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 the instructions in this documentation that are appropriate for the type of replication configuration that you will be using.

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.

If the source database will remain active during the initial load, include the HANDLECOLLISIONS parameter in the Replicat parameter file; otherwise do not use it. HANDLECOLLISIONS accounts for collisions that occur during the overlap of time between the initial load and the ongoing change replication. It reconciles insert operations for which the row already exists, and it reconciles update and delete operations for which the row does not exist. It can be used in these ways:

- globally for all tables in a parameter file
- as an on/off toggle for groups of tables
- within MAP statements to enable or disable the error handling for specific table pairs.

For more information about this parameter, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

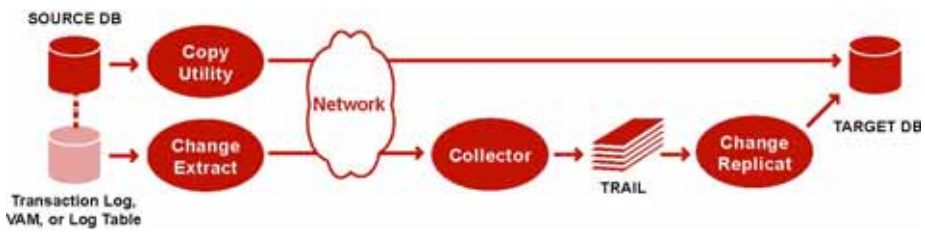
Sharing parameters between process groups

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.

For more information about using OBEY, see page 31.

For more information about macros, see page 233.

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.

To load data with a database utility

1. Make certain that you have addressed the requirements in "Prerequisites for initial load" on page 201.
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 name>
   ```

   Where: <group name> 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.

   WARNING 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 name>
```

Where: <group name> 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 name>
```

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 <Replicat group name>, NOHANDLECOLLISIONS
```

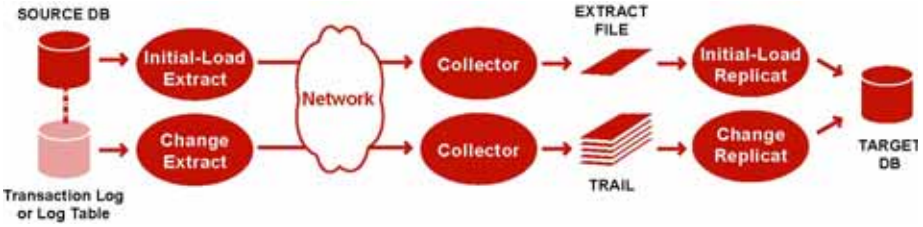
13. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

WARNING 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 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" on page 201.
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 name>
   ```

4. Enter the parameters listed in Table 26 in the order shown, starting a new line for each parameter statement.

Table 26 Initial-load Extract parameters for loading data from file to Replicat

Parameter	Description
`SOURCEISTABLE`	Designates Extract as an initial load process extracting records directly from the source tables.

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption information, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
PASSWORD is not required for NonStop SQL/MX or DB2.
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
`[, ENCRYPT <algorithm> KEYNAME <keyname>]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`ENCRYPTTRAIL <encryption options>`	Encrypts the data in the remote file. For encryption options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Table 26 Initial-load Extract parameters for loading data from file to Replicat (continued)

Parameter	Description
`RMTFILE <path name>, [MAXFILES <number>, MEGABYTES <n>]` ◆ <path name> is the relative or fully qualified name of the file. ◆ MAXFILES creates a series of files that are aged as needed. Use if the file could exceed the operating system's file size limitations. ◆ MEGABYTES designates the size of each file.	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**: The size of an extract file cannot exceed 2GB.
`TABLE <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of the table or a group of tables defined with wildcards. To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.	Specifies a source table or tables for initial data extraction.

5. Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
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 name>
   ```

8. Enter the parameters listed in Table 27 in the order shown, starting a new line for each parameter statement.

Table 27 Initial-load Replicat parameters for loading data from file to Replicat

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.

Table 27 Initial-load Replicat parameters for loading data from file to Replicat (continued)

Parameter	Description

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ TARGETDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`DECRYPTTRAIL <encryption options>`	Decrypts the data in the input file. For encryption options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`EXTFILE <path name> |`
`EXTTRAIL <path name>`
◆ <path name> is the relative or fully qualified name of the file or trail.
◆ Use EXTTRAIL only if you used the MAXFILES option of the RMTFILE parameter in the Extract parameter file.
	Specifies the input extract file specified with the Extract parameter RMTFILE.

`{SOURCEDEFS <file name>} |`
`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 Chapter 13.

`MAP <owner>.<table>,`
`TARGET <owner>.<table>;`
◆ <owner> is the schema name.
◆ <table> is the name of a table or a wildcard definition for multiple tables. To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.
	Specifies a relationship between a source and target table or tables.

9. Enter any appropriate optional Replicat parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
10. Save and close the file.

11. On the source system, start change extraction.

```
START EXTRACT <Extract group name>
```

12. (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>]
```

13. (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>
```

14. 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
name>.prm reportfile <path name>
```

    Windows:

```
C:\> <GGS directory>\extract paramfile dirprm\<initial-load Extract
name>.prm reportfile <path name>
```

    Where: <initial-load Extract name> is the name of the initial-load Extract that you used when creating the parameter file, and <path name> is the relative or fully qualified name of the Extract report file.

15. 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.
16. Wait until the initial extraction is finished.
17. On the target system, start the initial-load Replicat.

    UNIX and Linux:

```
$ /<GGS directory>/replicat paramfile dirprm/<initial-load Replicat
name>.prm reportfile <path name>
```

    Windows:

```
C:\> <GGS directory>\replicat paramfile dirprm\<initial-load Replicat
name>.prm reportfile <path name>
```

    Where: <initial-load Replicat name> is the name of the initial-load Replicat that you used when creating the parameter file, and <path name> is the relative or fully qualified name of the Replicat report file.

18. 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.
19. On the target system, start change replication.

```
START REPLICAT <Replicat group name>
```

20. On the target system, issue the following command to verify the status of change replication.

```
INFO REPLICAT <Replicat group name>
```

21. 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.
22. On the target system, issue the following command to turn off the HANDLECOLLISIONS parameter and disable the initial-load error handling.

```
SEND REPLICAT <Replicat group name>, NOHANDLECOLLISIONS
```

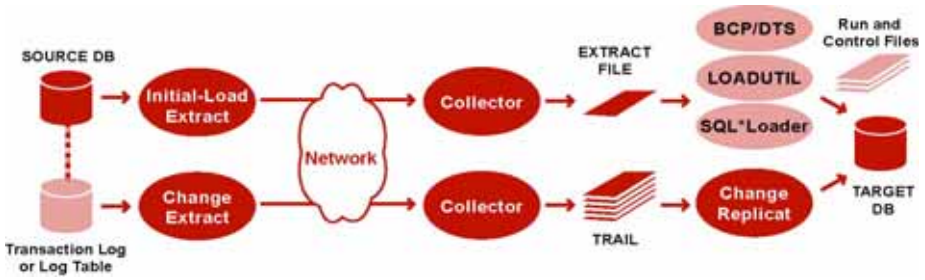
23. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

WARNING 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 from file to database utility



To use a database bulk-load utility, you use an initial-load Extract to extract source records from the source tables and write them to an extract file in external ASCII format. The file can be read by Oracle's SQL*Loader, Microsoft's BCP, DTS, or SQL Server Integration Services (SSIS) utility, or IBM's Load Utility (LOADUTIL). During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load. As part of the load procedure, Oracle GoldenGate uses the initial-load Replicat to create run and control files required by the database utility.

Any data transformation must be performed by the initial-load Extract on the source system because the control files are generated dynamically and cannot be pre-configured with transformation rules.

To load data from file to database utility

1. Make certain to satisfy "Prerequisites for initial load" on page 201.
2. On the source and target systems, run GGSCI and start Manager.

   ```
   START MANAGER
   ```

3. On the source system, issue the following command to create an initial-load Extract parameter file.

   ```
   EDIT PARAMS <initial-load Extract name>
   ```

4. Enter the parameters listed in Table 28 in the order shown, starting a new line for each parameter statement.

Table 28 Initial-load Extract parameters for loading from file to database utility

Parameter	Description
`SOURCEISTABLE`	Designates Extract as an initial load process that extracts records directly from the source tables.

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
PASSWORD is not required for NonStop SQL/MX or DB2.
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
`[, ENCRYPT <algorithm> KEYNAME <keyname>]`
`[, PARAMS - E -d <defs file>]`
◆ -E converts ASCII to EBCDIC.
◆ -d <defs file> specifies the source definitions file.
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.
	The PARAMS clause is necessary when loading with IBM's Load Utility, because Oracle GoldenGate will need to refer to the source definitions file.

`ENCRYPTTRAIL <encryption options>`	Encrypts the data in the remote file. For encryption options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Table 28 Initial-load Extract parameters for loading from file to database utility (continued)

Parameter	Description
`RMTFILE <path name>, [MAXFILES <number>, MEGABYTES <n>]` ◆ <path name> is the relative or fully qualified name of the file ◆ MAXFILES creates a series of files that are aged as needed. Use if the file could exceed the operating system's file size limitations. ◆ MEGABYTES designates the size of each file, up to 2MB.	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.
`FORMATASCII, {BCP | SQLLOADER}` ◆ BCP is used for BCP, DTS, or SSIS. ◆ SQLLOADER is used for Oracle SQL*Loader or IBM Load Utility.	Directs output to be formatted as ASCII text rather than the default canonical format. For information about limitations and options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.
`TABLE <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of the table or a group of tables defined with wildcards. To exclude tables from a wildcard specification, use the `TABLEEXCLUDE` parameter.	Specifies a source table or tables for initial data extraction.

5. Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
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 name>
   ```

8. Enter the parameters listed in Table 29 in the order shown, starting a new line for each parameter statement.

Table 29 Initial-load Replicat parameters for loading from file to database utility

Parameter	Description
`GENLOADFILES <template file>`	Generates run and control files for the database utility. For instructions on using this parameter, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Table 29 Initial-load Replicat parameters for loading from file to database utility (continued)

Parameter	Description

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw> [encryption options>]]]`
◆ TARGETDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`DECRYPTTRAIL <encryption options>`	Decrypts the data in the input extract file. For encryption options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`EXTFILE <path name> |`
`EXTTRAIL <path name>`
◆ <path name> is the relative or fully qualified name of the file
◆ Use EXTTRAIL only if you used the MAXFILES option of the RMTFILE parameter in the Extract parameter file.
	Specifies the extract file specified with the Extract parameter RMTFILE.

`{SOURCEDEFS <path name>} |`
`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 Chapter 13.

`MAP <owner>.<table>,`
`TARGET <owner>.<table>;`
◆ <owner> is the schema name.
◆ <table> is the name of a table or a wildcard definition for multiple tables. To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.
	Specifies a relationship between a source and target table or tables.

9. Enter any appropriate optional Replicat parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
10. Save and close the parameter file.

11. On the source system, start change extraction.

```
START EXTRACT <Extract group name>
```

12. (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>]
```

13. (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>
```

14. 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
name>.prm reportfile <path name>
```

    Windows:

```
C:\> <GGS directory>\extract paramfile dirprm\<initial-load Extract
name>.prm reportfile <path name>
```

    Where: <initial-load Extract name> is the name of the initial-load Extract that you used when creating the parameter file, and <path name> is the relative or fully qualified name of the Extract report file.

15. 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.
16. Wait until the initial extraction is finished.
17. On the target system, start the initial-load Replicat.

    UNIX and Linux:

```
$ /<GGS directory>/replicat paramfile dirprm/<initial-load Replicat
name>.prm reportfile <path name>
```

    Windows:

```
C:\> <GGS directory>\replicat paramfile dirprm\<initial-load Replicat
name>.prm reportfile <path name>
```

    Where: <initial-load Replicat name> is the name of the initial-load Replicat that you used when creating the parameter file, and <path name> is the relative or fully qualified name of the Replicat report file.

18. 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.
19. Using the ASCII-formatted extract files and the run and control files created by the initial-load Replicat, load the data with the database utility.

20. Wait until the load into the target tables is complete.
21. On the target system, start change replication.

```
START REPLICAT <Replicat group name>
```

22. On the target system, issue the following command to verify the status of change replication.

```
INFO REPLICAT <group name>
```

23. 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.
24. On the target system, issue the following command to turn off the HANDLECOLLISIONS parameter and disable the initial-load error handling.

```
SEND REPLICAT <Replicat group name>, NOHANDLECOLLISIONS
```

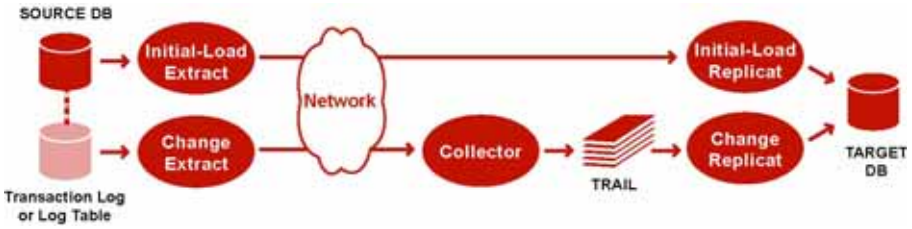
25. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

WARNING 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..

26. 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.

NOTE This method does not support extraction of LOB or LONG data. As an alternative, see "Loading data from file to Replicat" on page 204 or "Loading data from file to database utility" on page 209.

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.

Oracle GoldenGate direct load does not support tables that have columns that contain LOBs, LONGs, user-defined types (UDT), or any other large data type that is greater than 4 KB in size.

To load data with an Oracle GoldenGate direct load

1. Make certain to satisfy "Prerequisites for initial load" on page 201.
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 name>, SOURCEISTABLE
   ```

   Where:

   - <initial-load Extract name> 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 name>
   ```

5. Enter the parameters listed in Table 30 in the order shown, starting a new line for each parameter statement.

Table 30 Initial-load Extract parameters for Oracle GoldenGate direct load

Parameter	Description
`EXTRACT <initial-load Extract name>`	Specifies the initial-load Extract that you created in step 3.

Table 30 Initial-load Extract parameters for Oracle GoldenGate direct load (continued)

Parameter	Description

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
PASSWORD is not required for NonStop SQL/MX or DB2.
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
`[, ENCRYPT <algorithm> KEYNAME <keyname>]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`RMTTASK replicat,`
`GROUP <initial-load Replicat name>`
◆ <initial-load Replicat name> is the name of the initial-load Replicat group
	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.

`TABLE <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of the table or a group of tables defined with wildcards. To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.	Specifies a source table or tables for initial data extraction.

6. Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
7. Save and close the file.
8. On the target system, issue the following command to create the initial-load Replicat task.

   ```
   ADD REPLICAT <initial-load Replicat name>, SPECIALRUN
   ```

   Where:

   - <initial-load Replicat name> is the name of the initial-load Replicat task.
   - SPECIALRUN identifies the initial-load Replicat as a one-time run, not a continuous process.

9. On the target system, issue the following command to create an initial-load Replicat parameter file.

   ```
   EDIT PARAMS <initial-load Replicat name>
   ```

10. Enter the parameters listed in Table 31 in the order shown, starting a new line for each parameter statement.

Table 31 Initial-load Replicat parameters for Oracle GoldenGate direct load

Parameter	Description
`REPLICAT <initial-load Replicat name>`	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat in step 8.

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ TARGETDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption information, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`{SOURCEDEFS <full_pathname>} |`
`ASSUMETARGETDEFS`
◆ 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.
	Specifies how to interpret data definitions. For more information about data definitions files, see Chapter 13.

`MAP <owner>.<table>,`
`TARGET <owner>.<table>;`
◆ <owner> is the schema name.
◆ <table> is the name of a table or a wildcard definition for multiple tables. To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.
	Specifies a relationship between a source and target table or tables.

11. Enter any appropriate optional Replicat parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
12. Save and close the parameter file.
13. On the source system, start change extraction.

    ```
    START EXTRACT <Extract group name>
    ```

14. (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>]
    ```

15. (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>
    ```

16. On the source system, start the initial-load Extract.

    ```
    START EXTRACT <initial-load Extract name>
    ```

    NOTE Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

17. 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.

    ```
    VIEW REPORT <initial-load Extract name>
    ```

18. On the target system, start change replication.

    ```
    START REPLICAT <Replicat group name>
    ```

19. On the target system, issue the following command to verify the status of change replication.

    ```
    INFO REPLICAT <Replicat group name>
    ```

20. 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.
21. On the target system, issue the following command to turn off the HANDLECOLLISIONS parameter and disable the initial-load error handling.

    ```
    SEND REPLICAT <Replicat group name>, NOHANDLECOLLISIONS
    ```

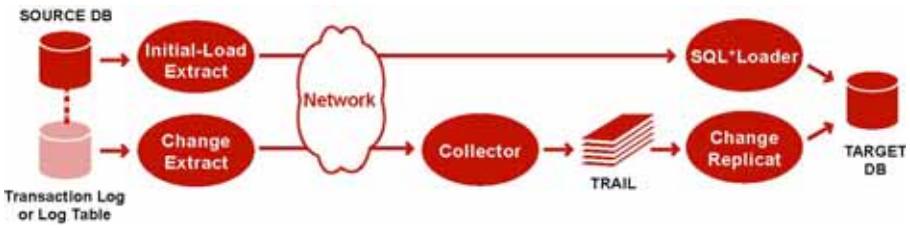
22. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

    WARNING 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..

23. 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.

Limitations:

- This method only works with Oracle's SQL*Loader. Do not use it for other databases.
- This method does not support extraction of LOB or LONG data. As an alternative, see "Loading data from file to Replicat" on page 204 or "Loading data from file to database utility" on page 209.
- This method does not support materialized views that contain LOBs, regardless of their size. It also does not support data encryption.

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" on page 201.
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 name>, SOURCEISTABLE
   ```

   Where:

   - <initial-load Extract name> 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 name>
   ```

6. Enter the parameters listed in Table 32 in the order shown, starting a new line for each parameter statement.

Table 32 Initial-load Extract parameters for a direct bulk load to SQL*Loader

Parameter	Description
`EXTRACT <initial-load Extract name>`	Specifies the initial-load Extract that you created in step 4.

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>`
`[<encryption options>]]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption information, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
PASSWORD is not required for NonStop SQL/MX or DB2.
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
`[, ENCRYPT <algorithm> KEYNAME <keyname>]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`RMTTASK replicat,`
`GROUP <initial-load Replicat name>`
◆ <initial-load Replicat name> is the name of the initial-load Replicat group.
	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.

Table 32 Initial-load Extract parameters for a direct bulk load to SQL*Loader (continued)

Parameter	Description
`TABLE <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of the table or a group of tables defined with wildcards. To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.	Specifies a table or tables for initial data extraction.

7. Enter any appropriate optional parameters.
8. Save and close the file.

9. On the target system, issue the following command to create the initial-load Replicat.

   ```
   ADD REPLICAT <initial-load Replicat name>, SPECIALRUN
   ```

   Where:

   - <initial-load Replicat name> is the name of the initial-load Replicat task.
   - SPECIALRUN identifies the initial-load Replicat as a one-time task, not a continuous process.

10. On the target system, issue the following command to create an initial-load Replicat parameter file.

    ```
    EDIT PARAMS <initial-load Replicat name>
    ```

11. Enter the parameters listed in Table 33 in the order shown, starting a new line for each parameter statement.

Table 33 Initial-load Replicat parameters for direct load to SQL*Loader

Parameter	Description
`REPLICAT <initial-load Replicat name>`	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat in step 9.
`USERID <user>, PASSWORD <password> [<encryption options>]`	Specifies the user ID, password, and encryption options to be used by the initial-load Replicat for connecting to the Oracle target database. You can include a host string, for example: `USERID ogg@ora1.ora, PASSWORD & AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, & AES128 KEYNAME mykey1` This parameter also allows for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.
`BULKLOAD`	Directs Replicat to interface directly with the Oracle SQL*Loader interface.
`{SOURCEDEFS <full_pathname>} | ASSUMETARGETDEFS` ◆ 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.	Specifies how to interpret data definitions. For more information about data definitions files, see Chapter 13.

Table 33 Initial-load Replicat parameters for direct load to SQL*Loader (continued)

Parameter	Description
`MAP <owner>.<table>, TARGET <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of a table or a wildcard definition for multiple tables. To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.	Specifies a relationship between a source and target table or tables.

12. Enter any appropriate optional Replicat parameters listed in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
13. Save and close the parameter file.
14. On the source system, start change extraction.

    ```
    START EXTRACT <Extract group name>
    ```

15. (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>]
    ```

16. (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>
    ```

17. On the source system, start the initial-load Extract.

    ```
    START EXTRACT <initial-load Extract name>
    ```

    WARNING Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

18. 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 name>
    ```

19. On the target system, start change replication.

    ```
    START REPLICAT <Replicat group name>
    ```

20. On the target system, issue the following command to verify the status of change replication.

    ```
    INFO REPLICAT <Replicat group name>
    ```

21. 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.
22. On the target system, issue the following command to turn off the HANDLECOLLISIONS parameter and disable the initial-load error handling.

    ```
    SEND REPLICAT <Replicat group name>, 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.

    WARNING 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 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 Chapter 14.

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 <Extract group name>
   ```

3. Start the data pump(s), if used.

   ```
   START EXTRACT <data pump group name>
   ```

4. Open the Replicat parameter file(s) for editing.

   WARNING 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.

CHAPTER 16

Customizing Oracle GoldenGate processing

Overview of custom processing

The following features can help you to customize and streamline processing:

- 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.

Processing that can be performed 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.

Databases and data types that are supported by SQLEXEC

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters.

- Numeric data types

- Date data types
- Character data types

How you can use 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.

To execute a procedure within a TABLE or MAP statement

Syntax

```
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.
`<sp name>`	Specifies the name of the stored procedure to execute.
`ID <logical name>`	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 <param spec> | NOPARAMS`	Specifies whether or not the procedure accepts parameters. One of these options must be used (see "Using input and output parameters" on page 229).

To execute a query within a TABLE or MAP statement

Syntax

```
SQLEXEC (ID <logical name>, QUERY " <sql query> ",
{PARAMS <param spec> | NOPARAMS})
```

Argument	Description
`ID <logical name>`	Defines a logical name for the query. A logical name is required in order to extract values from the query results. ID <logical name> references the column values returned by the query.

Argument	Description
`QUERY " <sql query> "`	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 quotes and must be contained all on one line. To use quoted object names within a SQLEXEC query, the SQL query must be enclosed within single quotes, rather than double quotes, and the USEANSISQLQUOTES parameter must be used in the GLOBALS file to enforce SQL-92 rules for object and literal identifiers. The following is an example of using quoted object names in a query: `SQLEXEC 'SELECT "col1" from "schema"."table"'`
`PARAMS <param spec> | NOPARAMS`	Defines whether or not the query accepts parameters. One of these options must be used (see "Using input and output parameters" on page 229).

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 <procedure name>()"`	Execute a stored procedure
`SQLEXEC "<sql query>"`	Execute a query
`SQLEXEC "<database command>"`	Execute a database command

Argument	Description
`"call <procedure name> ()"`	Specifies the name of a stored procedure to execute. The statement must be enclosed within double quotes. Example: `SQLEXEC "call prc_job_count ()"`

Argument	Description
"<sql query>"	Specifies the name of a query to execute. The query must be contained all on one line and enclosed within double quotes. To use quoted object names within a SQLEXEC query, the SQL query must be enclosed within single quotes, rather than double quotes, and the USEANSISQLQUOTES parameter must be used in the GLOBALS file to enforce SQL-92 rules for object and literal identifiers. The following is an example of using quoted object names in a query: `SQLEXEC 'SELECT "col1" from "schema"."table"'` For best results, type a space after the begin quotes and before the end quotes.
"<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 the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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.

To pass input values

To pass data values to input parameters within a stored procedure or query, use the PARAMS option of SQLEXEC.

Syntax

```
PARAMS ([OPTIONAL | REQUIRED] <param name> = {<source column> | <GG 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 name> 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 name> entries are P1 and P2.

- {<source column> | <GG function} is the column or Oracle GoldenGate conversion function that provides input to the procedure.

To pass output values

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.

Example The following 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.

Contents of LOOKUP procedure:

```
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 The following examples implement the same logic as used in the previous example, but they execute a SQL query instead of a stored procedure and use the @GETVAL function in the column map.

NOTE Due to the space constraints of this document, the query appears on multiple lines. However, an actual query would have to be on one line. In addition, to break up an

Oracle GoldenGate parameter statement into multiple lines, an ampersand (&) line terminator is required.

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));
```

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));
```

NOTE Additional SQLEXEC options are available for use when a procedure or query includes parametes. See the full SQLEXEC documentation in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Handling errors

There are two types of error conditions to consider when implementing SQLEXEC:

1. The column map requires a column that is missing from the source database operation. This can occur for an update operation when the database uses compressed updates in the transaction log. 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.

   Or...

2. The database generates an error.

To handle 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.

To handle database errors

Use the ERROR option in the SQLEXEC clause to direct Oracle GoldenGate to respond in one of

the following ways:

Table 34 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 REPERROR parameter specified in the Replicat parameter file. Oracle GoldenGate continues processing other stored procedures or queries associated with the current TABLE or MAP statement before processing the error.
FINAL	Performs in a similar way to RAISE except that when an error associated with a procedure or query is encountered, any remaining stored procedures and queries are bypassed. Error processing is invoked immediately after the error.
FATAL	Causes Oracle GoldenGate to abend immediately upon encountering an error associated with a procedure or query.

Additional 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 parameter in the Extract parameter file or the TARGETDB and/or USERID 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.
- By default, output values are truncated at 255 bytes per parameter. If longer parameters are required, use the MAXVARCHARLEN option of SQLEXEC.
- When using a stored procedure or query that changes the target database, use the DBOP option in the SQLEXEC clause. DBOP ensures that the changes are committed to the database properly. Otherwise, they could potentially be rolled back.
- 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 DB2 documentation for more information.
- Do not use SQLEXEC for tables being processing by a data-pump Extract in pass-through mode.
- 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

By using Oracle GoldenGate macros in parameter files you can configure and reuse parameters, commands, and conversion functions. The following are some ways to use macros:

- Implementing multiple uses of a parameter statement.
- Consolidating multiple commands.
- Invoking other macros.
- Storing commonly used macros in a library.

Oracle GoldenGate macros work with the following parameter files:

- Manager
- Extract
- Replicat

Do not use macros to manipulate data for tables being processed by a data-pump Extract in pass-through mode.

Defining macros

To define an Oracle GoldenGate macro, use the MACRO parameter in the parameter file.

Syntax

```
MACRO #<macro name>
PARAMS (#<p1>, #<p2> [, ...])
BEGIN
<macro body>
END;
```

Table 35 Macro definition arguments

Argument	Description
MACRO	Required. Indicates an Oracle GoldenGate macro.

Table 35 Macro definition arguments

Argument	Description
`#<macro name>`	The name of the macro. See "Macro naming conventions" on page 234.
`PARAMS (#<p1>, #<p2>)`	The names of parameters. See "Macro naming conventions" on page 234. Parameter statements are optional. When using parameters, separate each parameter in a list with a comma, or 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 "Using input parameters" on page 235.
`BEGIN`	Begins the macro body. Must be specified before the macro body.

`<macro body>`	The macro body. A macro body can include any of the following types of statements.
	◆ Simple parameter statements, as in:
	`COL1 = COL2`
	◆ Complex statements, as in:
	`COL1 = #val2`
	◆ Invocations of other macros, as in:
	`#colmap (COL1, #sourcecol)`

`END`	Ends the macro definition.

Macro naming conventions

Observe the following naming conventions when creating macros and parameters:

- Macro and parameter names must begin with a macro character. The default macro character is the pound (#) character, as in #macro1 and #param1. Anything in the parameter file that begins with the # macro character is assumed to be either a macro or a macro parameter. Macro or parameter names within quotation marks are ignored.

  You can change the macro character to something other than #. For example, you could change it if table names include the # character. To define a different macro character, precede the MACRO statement with the MACROCHAR <character> parameter in the parameter file. In the following example, $ is defined as the macro character.

  ```
  MACROCHAR $
  MACRO $mymac
  PARAMS ($p1)
  BEGIN
  col = $p1
  END;
  ```

  The MACROCHAR parameter can only be used once.
- Macro and parameter names are not case-sensitive.
- Besides the leading macro character (# or user-defined), valid macro and parameter characters are alphanumeric and can include the underscore character (_).

Invoking a macro

To invoke a macro, place an invocation statement in the parameter file wherever you want the macro to occur.

Syntax

```
[<target> =] <macro name> (<val1>, <val2> [, ...])
```

Table 36 Macro invocation arguments

Argument	Description

`<target> =`	Optional. Specifies the target to which the results of the macro processing are assigned, typically a target column. For example, in the following, the column DATECOL1 is the target:
	`DATECOL1 = #make_date (YR1, MO1, DAY1)`
	Without a target, the syntax would be:
	`#make_date (YR1, MO1, DAY1)`

`<macro name>`	The name of the macro, such as: `#assign_date`

`(<val1>, <val2>)`	The values for parameters defined with a PARAMS statement in the macro definition, for example:
	`#custdate (#year, #month, #day)`
	If a macro does not require any parameters, then the parameter value list is empty, but the open and close parentheses still are required, for example:
	`#no_params_macro ()`
	Valid parameter values include plain text, quoted text, and invocations of other macros, as shown in the following examples.
	`my_col_1`
	`"your text here"`
	`#mycalc (col2, 100)`
	`#custdate (#year, #month, #day)`
	`#custdate (#getyyyy (#yy), #month, #day)`

Using input parameters

Using input parameters in a macro is optional.

To use macros with parameters

Use the PARAMS argument of the MACRO parameter to describe input parameters to the macro. Every parameter used in a macro must be declared in the PARAMS statement, and when the macro is invoked, the invocation must include a value for each parameter.

Example The following example illustrates how column mapping can be improved with a macro that uses parameters. In the following example, the 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 invoked as follows, with a value list in parentheses for each parameter:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcol1 = sourcecol1,
datecol1 = #make_date(YR1, MO1, DAY1),
datecol2 = #make_date(YR2, MO2, DAY2)
);
```

The macro expands to:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcol1 = sourcecol1,
datecol1 = @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)
);
```

To use macros without parameters

You can create macros without parameters. For example, you can create a macro for frequently used sets of commands, as in the following example.

Example

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

The macro is invoked as follows, without parameter values in the parentheses:

```
#option_defaults ()
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

#option_defaults ()
MAP owner.srctab2, TARGET owner.targtab2;
```

The macro expands to:

```
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
MAP owner.srctab2, TARGET owner.targtab2;
```

Parameter substitution

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 invocation is substituted.
3. If a parameter name does not appear in the list, the macro processor evaluates whether or not the item is, instead, an invocation of another macro. (See "Invoking other macros from a macro" on page 237.)

Invoking other macros from a macro

To invoke other macros from a macro, create a macro definition similar to the following:

```
MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

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 macros" on page 233, 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.

Example 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 in a parameter file

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.

Example

```
INCLUDE /ggs/dirprm/mdatelib.mac
REPLICAT rep
ASSUMETARGETDEFS
USERID ogg@ora1.ora, PASSWORD &
   AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
MAP fin.acct_tab, TARGET fin.account;
```

Suppressing report file listing

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.

Example

```
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 invoked, 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.

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.

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.

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 the Oracle GoldenGate *Windows and UNIX Reference Guide*.

> **NOTE** User exits are case-sensitive for database object names. Names are returned exactly as they are defined in the hosting database.

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.
2. In the source code, include the usrdecs.h file, which is located in the Oracle GoldenGate installation directory. This file contains type definitions, return status values, callback function codes, and a number of other definitions. 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.

Syntax

```
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. The parameter also accepts options to:
   - use a user exit with a data pump that is operating in pass-through mode
   - get before images for update operations
   - supply a startup string, such as a startup parameter

Syntax

```
CUSEREXIT <DLL or shared object name> <routine name>
[, PASSTHRU]
[, INCLUDEUPDATEBEFORES]
[, PARAMS "<startup string>”]
```

Example Example parameter file syntax, UNIX systems:

```
CUSEREXIT eruserexit.so MyUserExit
```

Example Example parameter file syntax, Windows systems:

```
CUSEREXIT eruserexit.dll MyUserExit
```

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. 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_passthru.c shows how the PASSTHRU option of the CUSEREXIT parameter can be used in an Extract data pump.
- 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 file as well as makefiles and a readme.txt file.

Upgrading your 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.

Using the Oracle GoldenGate event marker system to raise database events

Oracle GoldenGate provides an *event marker* system that 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.

   For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

You can combine EVENTACTIONS options, as shown in the following examples.

Example 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)
```

Example The following writes the event record to the discard file and ignores the entire transaction.

```
EVENTACTIONS (DISCARD, IGNORE TRANS)
```

Example The following logs an informational message and gracefully stop the process.

```
EVENTACTIONS (LOG INFO, STOP)
```

Example The following rolls over the trail file and does not write the event record to the new file.

```
EVENTACTIONS (ROLLOVER, IGNORE)
```

Case studies in the usage of the event marker system

These examples highlight some use cases for the event marker system. For syntax details, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Trigger end-of-day processing

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

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.

Simplify transition from initial load to change synchronization

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

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.

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

```
MAP source.account, TARGET target.account;
TABLE source.account, FILTER (withdrawal > balance), EVENTACTIONS (ABORT);
```

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.

Use a heartbeat table and logging to analyze lag

```
ALLOWDUPTARGETMAPS
MAP source.heartbeat, TARGET target.heartbeat, &
FILTER ( &
@DATEDIFF ("SS", hb_timestamp, @DATENOW()) > 60 AND &
@DATEDIFF ("SS", hb_timestamp, @DATENOW()) < 120), &
EVENTACTIONS (LOG INFO);

MAP source.heartbeat, TARGET target.heartbeat, &
FILTER (@DATEDIFF ("SS", hb_timestamp, @DATENOW()) > 120), &
EVENTACTIONS (LOG WARNING);
```

This example shows how to configure log actions. A heartbeat table is periodically updated with the current time in the source database. The updates to the heartbeat table are captured and written to the trail by Extract. Using the heartbeat record as the event record, Replicat logs messages based on lag calculations in two different MAP statements with FILTER clauses.

In the first FILTER clause, an informational message is written if the lag exceeds 60 seconds but is less than 120 seconds. In the second FILTER clause, a warning message is logged if the lag exceeds 120 seconds.

In this example, the heartbeat record is also written to a heartbeat audit table in the target database, which can be used for historical lag analysis. An alternate option would be to IGNORE or DISCARD the heartbeat record.

> **NOTE** ALLOWDUPTARGETMAPS is used because there are duplicate MAP statements for the same source and target objects.

Trace a specific order number

```
MAP sales.order, TARGET rpt.order;
TABLE source.order,
FILTER (@GETENV ("GGHEADER", "OPTYPE") = "INSERT" AND order_no = 1), &
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

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.

Execute a batch process

```
TABLE source.job, FILTER (@streq(job_type = "HOUSEKEEPING")=1), &
EVENTACTIONS (IGNORE TRANSACTION);
```

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.

> **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

Extract:

```
TABLE source.statement, EVENTACTIONS (IGNORE TRANS INCLUDEEVENT);
```

Replicat:

```
TABLE source.statement, SQLEXEC (<execute SQL statement>), &
EVENTACTIONS (INFO, IGNORE);
```

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.

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

```
TABLE source.batch_table, EVENTACTIONS (CHECKPOINT BEFORE);
```

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.

Execute a shell script to validate data

Extract:

```
TABLE src.*;
TABLE test.event;
```

Replicat:

```
MAP src.*, TARGET targ.*;
TABLE test.event, FILTER (@streq(event_type, "COMPARE")=1), &
EVENTACTIONS (SHELL "compare_db.sh", FORCESTOP);
```

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.

CHAPTER 17

Monitoring Oracle GoldenGate processing

Overview of the Oracle GoldenGate monitoring tools

You can monitor Oracle GoldenGate processing to view process status, statistics, and events by using the following tools.

- GGSCI information commands
- The ggserr.log file (known as the *error log*)
- Process reports
- The discard file
- The Event Viewer on Windows systems or the syslog on UNIX systems to view errors at the operating-system level.

Using the information commands in GGSCI

The primary way to view processing information is through GGSCI. For more information about these commands, see the *Oracle GoldenGate Windows and UNIX Reference Guide*.

Table 37 Commands to view process information

Command	What it shows
INFO {EXTRACT | REPLICAT} <group> [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} <group>	Statistics for operations processed
STATUS {EXTRACT | REPLICAT} <group>	Run status (starting, running, stopped, abended)
STATUS MANAGER	Run status
LAG {EXTRACT | REPLICAT} <group>	Latency between last record processed and timestamp in the data source

Table 37 Commands to view process information (continued)

Command	What it shows
INFO {EXTTRAIL | RMTTRAIL} <path name>	Name of associated process, position of last data processed, maximum file size
SEND MANAGER	Run status, information about child processes, port information, trail purge settings
SEND {EXTRACT | REPLICAT}	Depending on the process, returns information about memory pool, lag, TCP statistics, long-running transactions, process status, recovery progress, and more.
VIEW REPORT <group>	Contents of the process report
VIEW GGSEVT	Contents of the Oracle GoldenGate error log

<command> ER <wildcard>	Information dependent on the <command> type:
	INFO
	LAG
	SEND
	STATS
	STATUS
	
	<wildcard> is a wildcard specification for the process groups to be affected, for example:
	`INFO ER ext*`
	`STATS ER *`

Monitoring an Extract recovery

NOTE This topic applies to all types of databases except Oracle, for which a different recovery mechanism known as *Bounded Recovery* is used. For more information, see the BR parameter in the Oracle GoldenGate *Windows and UNIX Reference Guide*.

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.
- In recovery[2] – Extract is recovering from its checkpoint to the end of the trail.
- 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. For help with tuning Oracle GoldenGate to minimize lag, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

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.

Syntax
```
LAG {EXTRACT | REPLICAT | ER} {<group | wildcard>}
```

Or...

Syntax
```
SEND {EXTRACT | REPLICAT} {<group | wildcard>}, GETLAG
```

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.

Figure 19 Sample lag statistics for all Extract and Replicat processes

```
GGSCI (sysb) 13> lag er *

Sending GETLAG request to EXTRACT ORAEXT...
Last record lag: 1 seconds.
At EOF, no more records to process.

Sending GETLAG request to REPLICAT ORAREP...
No records yet processed.
At EOF, no more records to process.

Sending GETLAG request to REPLICAT REPORA...
Last record lag: 7 seconds.
At EOF, no more records to process.
```

To control how lag is reported

Use the LAGREPORTMINUTES or LAGREPORTHOURS parameter to specify the interval at which Manager checks for Extract and Replicat lag.

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.

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 LAGREPORTHOURS parameter.

Monitoring processing volume

The volume statistics 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.

To view volume statistics

Syntax

```
STATS {EXTRACT | REPLICAT | ER} {<group | wildcard>}
[TABLE {<name | wildcard>}]
```

Figure 20 Sample basic STATS EXTRACT for one table

```
GGSCI (sysa) 32> stats extract oraext
Sending STATS request to EXTRACT ORAEXT...
Start of Statistics at 2011-01-08 16:46:38.
Output to c:\goldengate802\dirdat\xx:
Extracting from HR.EMPLOYEES to HR.EMPLOYEES:

*** Total statistics since 2011-01-08 16:35:05 ***
        Total inserts                                704.00
        Total updates                                  0.00
        Total deletes                                160.00
        Total discards                                 0.00
        Total operations                             864.00

*** Daily statistics since 2011-01-08 16:35:05 ***
        Total inserts                                704.00
        Total updates                                  0.00
        Total deletes                                160.00
        Total discards                                 0.00
        Total operations                             864.00

*** Hourly statistics since 2011-01-08 16:35:05 ***
        Total inserts                                704.00
        Total updates                                  0.00
        Total deletes                                160.00
        Total discards                                 0.00
        Total operations                             864.00
```

```
*** Latest statistics since 2011-01-08 16:35:05 ***
        Total inserts                              704.00
        Total updates                                0.00
        Total deletes                              160.00
        Total discards                               0.00
        Total operations                           864.00
```

To view the processing rate

Syntax

```
STATS {EXTRACT | REPLICAT | ER} {<group | wildcard>},
REPORTRATE {HR | MIN | SEC}
```

Figure 21 Sample REPORTRATE output

```
*** Latest statistics since 2011-01-08 16:35:05 ***
        Total inserts/hour:                        718.34
        Total updates/hour:                          0.00
        Total deletes/hour:                          0.00
        Total discards/hour:                         0.00
        Total operations/hour:                     718.34
```

To view a summary of operations processed per table since startup

Syntax

```
STATS {EXTRACT | REPLICAT | ER} {<group | wildcard>},
TOTALSONLY <table>
```

Figure 22 Sample TOTALSONLY output

```
GGSCI (sysa) 37> stats extract oraext, totalsonly hr.departments

Sending STATS request to EXTRACT ORAEXT...
Start of Statistics at 2011-01-08 17:06:43.
Output to c:\goldengate802\dirdat\xx:
Cumulative totals for specified table(s):

*** Total statistics since 2011-01-08 16:35:05 ***
        Total inserts                              352.00
        Total updates                                0.00
        Total deletes                                0.00
        Total discards                               0.00
        Total operations                           352.00

*** Daily statistics since 2011-01-08 16:35:05 ***
        Total inserts                              352.00
        Total updates                                0.00
        Total deletes                                0.00
        Total discards                               0.00
        Total operations                           352.00

*** Hourly statistics since 2011-01-08 17:00:00 ***

        No database operations have been performed.
```

```
*** Latest statistics since 2011-01-08 16:35:05 ***
        Total inserts                              352.00
        Total updates                                0.00
        Total deletes                                0.00
        Total discards                               0.00
        Total operations                           352.00

End of Statistics.
```

To limit the types of statistics that are displayed

Syntax

```
STATS {EXTRACT | REPLICAT | ER} {<group | wildcard>},
{TOTAL | DAILY | HOURLY | LATEST}
```

Figure 23 Sample LATEST statistics

```
GGSCI (sysa) 39> stats extract oraext, latest

Sending STATS request to EXTRACT ORAEXT...
Start of Statistics at 2011-01-08 17:18:23.
Output to c:\goldengate802\dirdat\xx:
Extracting from HR.EMPLOYEES to HR.EMPLOYEES:

*** Latest statistics since 2011-01-08 16:35:05 ***
        Total inserts                              704.00
        Total updates                                0.00
        Total deletes                              160.00
        Total discards                               0.00
        Total operations                           864.00

End of Statistics.
```

To clear all filters that were set with previous options

Syntax

```
STATS {EXTRACT | REPLICAT | ER} {<group | wildcard>}, RESET
```

To send interim statistics to the report file

Syntax

```
SEND {EXTRACT | REPLICAT | ER} {<group | wildcard>}, REPORT
```

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 ggserr.log file within the root Oracle GoldenGate directory
- Oracle GoldenGate Director
- VIEW GGSEVT command in GGSCI

Syntax

```
VIEW GGSEVT
```

To filter the error log

The error log can become very large, but you can filter it based on a keyword. For example, this filter show only errors:

```
$ more ggserr.log | grep ERROR
```

Because the error log will continue to grow as you use Oracle GoldenGate, consider archiving and deleting the oldest entries in the file.

NOTE The Collector process might stop reporting to the log on UNIX systems after the log has been cleaned up. To get reporting started again, restart the Collector process after the cleanup.

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 at the end of each run. 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 Director
- VIEW REPORT command in GGSCI

Syntax

```
VIEW REPORT {<group> | <file name> | MGR}
```

Where:

- <group> shows an Extract or Replicat report that has the default name, which is the name of the associated group.
- <file name> shows any Extract or Replicat report file that matches a given path name. Must be used if a non-default report name was assigned with the REPORT option of the ADD EXTRACT or ADD REPLICAT command when the group was created.
- MGR shows the Manager process report.

Report names are in upper case if the operating system is case-sensitive. 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.

To determine the name and location of a process report

Use the INFO command in GGSCI.

Syntax

```
INFO <group>, DETAIL
```

To view information if a process abends without a report

Run the process from the command shell of the operating system (not GGSCI) to send the information to the terminal.

Syntax

```
<process> paramfile <path name>.prm
```

Where:

- <process> is either Extract or Replicat.
- paramfile <path name>.prm is the fully qualified name of the parameter file.

Example

```
replicat paramfile /ggs/dirdat/repora.prm
```

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.

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.

Managing process reports

Once created, a report file must remain in its original location for Oracle GoldenGate to operate properly after processing has started.

Whenever a process starts, Oracle GoldenGate creates a new report file and ages the previous one by appending a sequence number to the name. The numbers increment from 0 (the previous one) to 9 (the oldest).

No process ever has more than ten aged reports and one active report. After the tenth aged report, the oldest is deleted when a new report is created. Set up an archiving schedule for aged report files in case they are needed to resolve a service request.

Figure 24 Current Extract and Manager reports plus aged reports

```
-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
-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
```

To prevent an Extract or Replicat report file from becoming too large

Use the REPORTROLLOVER parameter to force report files to age on a regular schedule, instead of when a process starts. For long or continuous runs, setting an aging schedule controls the size of the active report file and provides a more predictable set of archives that can be included in your archiving routine.

To prevent SQL errors from filling up the Replicat report

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.

Using the discard file

Use a discard file to capture information about Oracle GoldenGate operations that failed. This information can help you to 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.

A discard file can be used for Extract or Replicat, but it is most useful for Replicat to log operations that could not be reconstructed or applied.

To use a discard file

Include the DISCARDFILE parameter in the Extract or Replicat parameter file. You must supply a name for the file. The parameter has options that control the maximum file size, after which the process abends, and whether new content overwrites or appends to existing content.

Syntax

```
DISCARDFILE <file name> [, APPEND | PURGE] [, MAXBYTES <n> | MEGABYTES <n>]
```

NOTE To prevent the need to perform manual maintenance of discard files, use either the PURGE or APPEND option. Otherwise, you must specify a different discard file name before starting each process run, because Oracle GoldenGate will not write to an existing discard file.

To view a discard file

Use either of the following:

- Standard shell command to view the file by name
- VIEW REPORT command in GGSCI, with the discard file name as input

Syntax

```
VIEW REPORT <file name>
```

To manage discard files

Use the DISCARDROLLOVER parameter to set a schedule for aging discard files. For long or continuous runs, setting an aging schedule prevents the discard file from filling up and causing the process to abend, and it provides a predictable set of archives that can be included in your archiving routine.

Syntax

```
DISCARDROLLOVER {AT <hh:mi> | ON <day of week> | AT <hh:mi> ON <day of week>}
```

Using the system logs

Oracle GoldenGate writes errors that are generated at the level of the operating system to the Event Viewer on Windows or to the syslog on UNIX and Linux. Oracle GoldenGate events are basically the same format in the UNIX, Linux, and Windows system logs. Oracle GoldenGate errors that appear in the system logs also appear in the Oracle GoldenGate error log.

On UNIX and Linux, Oracle GoldenGate messaging to the syslog is enabled by default. On Windows, Oracle GoldenGate messaging to the Event Viewer must be installed by registering the Oracle GoldenGate message DLL.

To register Oracle GoldenGate messaging on Windows

1. Run the install program with the addevents option. This enables generic messages to be logged.
2. (Optional) To get more specific Windows messages, copy the category.dll and ggsmsg.dll libraries from the Oracle GoldenGate directory to the SYSTEM32 directory, either before or after running install. The detailed messages contain the Oracle GoldenGate user name and process, the name of the parameter file, and the error text.

   NOTE Windows event messaging might have been installed when Oracle GoldenGate was installed. For more information on running install, see the Oracle GoldenGate installation guide for your database.

To filter Oracle GoldenGate messaging on Windows and UNIX

Use the SYSLOG parameter to control the types of messages that Oracle GoldenGate sends to the system logs on a Windows or UNIX system. You can:

- include all Oracle GoldenGate messages
- suppress all Oracle GoldenGate messages
- filter to include information, warning, or error messages, or any combination of those types

You can use SYSLOG as a GLOBALS or Manager parameter, or both. When present in the GLOBALS parameter file, it controls message filtering for all of the Oracle GoldenGate processes on the system. When present in the Manager parameter file, it controls message filtering only for the Manager process. If used in both the GLOBALS and Manager parameter files, the Manager setting overrides the GLOBALS setting for the Manager process. This enables you to use separate settings for Manager and all of the other Oracle GoldenGate processes.

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.

Sending event messages to a NonStop system

Event messages created by Collector and Replicat processes on a Windows or UNIX system can be captured and sent to EMS on NonStop systems. This feature enables centralized viewing of Oracle GoldenGate messages across platforms. To use this feature, there are two procedures:

- Run the EMS client on the Windows or UNIX system
- Start a Collector process on the NonStop system

Running EMSCLNT on a Windows or UNIX system

The EMSCLNT utility captures Oracle GoldenGate event messages that originate on a Windows or UNIX system and sends them to a TCP/IP Collector process on the NonStop

system. EMSCLNT reads a designated error log and runs indefinitely, waiting for more messages to send.

Run emsclnt from the Oracle GoldenGate directory on the Windows or UNIX system, using the following syntax:

```
emsclnt -h <hostname> | <IP address>
-p <port number>
-f <filename>
-c <Collector>
```

Where:

- -h <hostname> | <IP address> is either the name or IP address of the NonStop server to which EMS messages will be sent.
- -p <port number> is the port number of the NonStop Collector process.
- -f <filename> is the name of the local file from which to distribute error messages. Use the full path name if the file resides somewhere other than the Oracle GoldenGate directory.
- -c <Collector> is the EMS Collector for this client.

Example The following Windows example, executed from the DOS prompt, reads the file d:\ggs\ggserr.log for error messages. Error messages are sent to the Collector on NonStop host ggs2 listening on port 9876. The Collector process on NonStop writes formatted messages to EMS Collector $0.

```
> emsclnt –h ggs2 –p 9876 –f d:\ggs\ggserr.log –c $0
```

Example The following UNIX example reads the file ggserr.log for error messages. Error messages are sent to the Collector on the NonStop server at IP address 13.232.123.89 listening on port 7850. The Collector on NonStop writes formatted messages to EMS Collector $0.

```
emsclnt –h 13.232.123.89 –p 7850 –f ggserr.log –c '$0'
```

NOTE Because the dollar sign on UNIX denotes a variable, the $0 must be within single quotes.

Running the Collector on NonStop

The Collector on the NonStop system (called Server-Collector on that platform) collects and distributes the EMS messages. To start the Collector, run the server program. For each EMSCLNT process that you will be running on a Windows or UNIX system, start one server process on the NonStop system.

For example, the following runs server and outputs messages to $DATA1.GGSERRS.SERVLOG.

```
> ASSIGN STDERR, $DATA1.GGSERRS.SERVLOG
> RUN SERVER /NOWAIT/ –p 7880
```

See the Oracle GoldenGate *HP NonStop Reference Guide* for details about running a Server-Collector process on the NonStop platform.

Getting more help with monitoring and tuning

For more information about how to monitor, tune, and troubleshooting Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

CHAPTER 18

Performing administrative operations

Overview of 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. The procedures that are covered are:

- Performing application patches
- Adding process groups
- Initializing the transaction logs
- Shutting down the system
- Changing database attributes
- Changing the size of trail files

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 page 269.
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.

Adding process groups

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 a parallel Extract group to an active configuration
- Adding a data pump to an active configuration
- Adding a parallel Replicat group to an active configuration

Adding a parallel Extract group to an active configuration

This procedure adds a new Extract group in parallel to an existing Extract group. 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. 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;
```

10. Unlock the table(s) if you did not use the script in the previous step.
11. Stop the old Extract group(s) and any existing data pumps.

```
STOP EXTRACT <group>
```

12. 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>
```

13. Add a trail for the new Extract group.

```
ADD {EXTTRAIL | RMTTRAIL} <trail name>, 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 "To add a local data pump".
- 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 "To add a remote Replicat".

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 add a local data pump

1. On the source system, add the data-pump Extract group using the EXTTRAIL trail as the data source.

```
ADD EXTRACT <group>, EXTTRAILSOURCE <trail name>
```

For example:

```
ADD EXTRACT pump, EXTTRAILSOURCE dirdat\lt
```

2. Create a parameter file for the data pump.

```
EDIT PARAMS <group>
```

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.

     NOTE If the data pump will be pumping data, but not performing filtering, mapping, or conversion, then you can include the PASSTHRU parameter to bypass the overhead of database lookups. You also can omit database authentication parameters.

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 name>, EXTRACT <group>
   ```

   For example:

   ```
   ADD RMTTRAIL dirdat/rt, EXTRACT pump
   ```

5. Follow the steps in "To add a remote Replicat"

To add a remote Replicat

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 rep, 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.

   ```
   STOP EXTRACT <group>
   START EXTRACT <group>
   ```

4. On the target system, start the new Replicat group.

   ```
   START REPLICAT <group>
   ```

Adding a 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 name>, 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.

   WARNING 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 name>
   ```

   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).

     NOTE If the data pump will be pumping data, but not performing filtering, mapping, or conversion, you can include the PASSTHRU parameter to bypass the overhead of database lookups. You also can omit database authentication parameters.

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 name>, 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 <dsn>] [, USERID <user>[, PASSWORD <password>
[<encryption 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 a parallel Replicat group to an active configuration

This procedure adds a new Replicat group in parallel to an existing Replicat group. The

new Replicat reads from the same trail as the original Replicat.

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 moving to this group.
5. Save and close the parameter file.
6. On the source system, run GGSCI.
7. Stop the Extract group.

   ```
   STOP EXTRACT <group>
   ```

8. On the target system, edit the old Replicat parameter file to remove the 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.
9. Save and close the file.
10. Issue the LAG REPLICAT command for the old Replicat group, and continue issuing it until it reports "At EOF, no more records to process."

    ```
    LAG REPLICAT <group>
    ```

11. Stop the old Replicat group.

    ```
    STOP REPLICAT <group>
    ```

12. 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
    ```

13. On the source system, start the Extract group.

    ```
    START EXTRACT <group>
    ```

14. On the target system, start the old Replicat group.

    ```
    START REPLICAT <group>
    ```

15. Start the new Replicat group.

```
START REPLICAT <group>
```

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 name> 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 name> 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 name>
STOP EXTRACT <pump name>
STOP REPLICAT <group name>
```

7. Delete the Extract, data pump, and Replicat groups.

```
DELETE EXTRACT <group name>
DELETE EXTRACT <pump name>
DELETE REPLICAT <group name>
```

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 name> TRANLOG, BEGIN NOW
```

12. Recreate the local trail (if used).

```
ADD EXTTRAIL <trail name>, EXTRACT <group name>
```

13. Recreate the data pump (if used).

```
ADD EXTRACT <pump name>, EXTTRAILSOURCE <local trail name>
```

14. Recreate the remote trail.

```
ADD RMTTRAIL <trail name>, EXTRACT <pump name>
```

15. Recreate the Replicat group.

```
ADD REPLICAT <group name>, EXTTRAIL <trail name>
```

16. Start Extract, the data pump (if used), and Replicat.

```
START EXTRACT <group name>
START EXTRACT <pump name>
START REPLICAT <group name>
```

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 name> 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.

Adding tables to the source database

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 Teradata 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

- 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 the Oracle GoldenGate *Windows and UNIX Reference Guide* 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 name>
   ```

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 definitins 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 name>
   START REPLICAT <group name>
   ```

9. Start Extract, if applicable.

   ```
   START EXTRACT <group name>
   ```

10. Permit user access to the new tables.

Coordinating table attributes between source and target

NOTE See also “Performing an ALTER TABLE to add a column on DB2 z/OS tables”

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 This procedure assumes that the Oracle GoldenGate DDL support feature is not in use, or is not supported for your database. For Oracle and Teradata 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.

WARNING 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 <data pump>
START EXTRACT <data pump>
```

9. On the target system, run GGSCI.
10. Open the Replicat parameter file for editing.

    WARNING 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 name>
   ```

3. Stop Extract.

   ```
   STOP EXTRACT <group name>
   ```

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 redo threads

Any time the number of redo threads in an Oracle RAC database cluster changes, the Extract group must be dropped and re-added. 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 name>
STOP REPLICAT <group name>
```

3. On the source system, issue the following command to delete the primary Extract group and the data pump.

```
DELETE EXTRACT <group name>
DELETE EXTRACT <pump name>
```

4. On the target system, issue the following command to delete the Replicat groups.

```
DELETE REPLICAT <group name>
```

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 name> TRANLOG, THREADS <n>, BEGIN NOW
```

7. Add the local trail again with the same name as before.

```
ADD EXTTRAIL <trail name>, EXTRACT <group name>
```

8. Add the data pump Extract again, with the same name as before.

```
ADD EXTRACT <group name> EXTTRAILSOURCE <trail name>, BEGIN NOW
```

9. Add the remote trail again with the same name as before.

```
ADD RMTTRAIL <trail name>, EXTRACT <group name>
```

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 name> EXTTRAIL <trail name>
```

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 name>
START REPLICAT <group name>
```

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 name>, 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.

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 name>, EXTRACT <group name>, MEGABYTES <n>
   ```

   (Local trail)

   ```
   ALTER EXTTRAIL <trail name>, EXTRACT <group name>, MEGABYTES <n>
   ```

3. Issue the following command to cause Extract to switch to the next file in the trail.

   ```
   SEND EXTRACT <group name>, ROLLOVER
   ```

CHAPTER 19

Undoing data changes with the Reverse utility

Overview of the Reverse utility

The Reverse utility uses before images to undo database changes for specified tables, records, and time periods. It enables you to perform a selective backout, unlike other methods which require restoring the entire database.

You can use the Reverse utility for the following purposes:

- To restore a test database to its original state before the test run. Because the Reverse utility only backs out changes, a test database can be restored in a matter of minutes, much more efficiently than a complete database restore, which can take hours.
- To reverse errors caused by corrupt data or accidental deletions. For example, if an UPDATE or DELETE command is issued without a WHERE clause, the Reverse utility reverses the operation.

To use the Reverse utility, you do the following:

- Run Extract to extract the before data.
- Run the Reverse utility to perform the reversal of the transactions.
- Run Replicat to apply the restored data to the target database.

The Reverse utility reverses the forward operations by:

- Reversing the ordering of database operations in an extract file, a series of extract files, or a trail so that they can be processed in reverse order, guaranteeing that records with the same key are properly applied.
- Changing delete operations to inserts.
- Changing inserts to deletes.
- Changing update before images to update after images.
- Reversing the begin and end transaction indicators.

Figure 25 Reverse utility architecture

Reverse utility restrictions

- Commit timestamps are not changed during the reverse procedure, which causes the time sequencing in the trail to be backwards. Because of this, you cannot position Replicat based on a timestamp.
- Oracle GoldenGate does not store the before images of the following data types, so these types are not supported by the Reverse utility. A before image is required to reverse update and delete operations.

Table 38 Data types not supported by the Reverse utility

DB2 (all supported OS)	Oracle	SQL Server	Sybase	Teradata
BLOB CLOB DBCLOB	CLOB BLOB NCLOB LONG LONG RAW XMLType UDT Nested Tables VARRAY	TEXT IMAGE NTEXT VARCHAR (MAX)	VARBINARY BINARY TEXT IMAGE	None supported. This is because only the after images of a row are captured by the Teradata vendor access module.

Configuring the Reverse utility

The reversal process is performed with online Extract and Replicat processes that are created and started through GGSCI. These processes write to standard local or remote trails. Oracle GoldenGate automatically reverses the file order during reverse processing so that transaction sequencing is maintained.

To configure the Reverse utility

To configure the Reverse utility, create Extract and Replicat parameter files with the parameters shown in Table 39 and Table 40. In addition to these parameters, include any other optional parameters or special MAP statements that are required for your synchronization configuration.

Table 39 Extract parameter file for the Reverse utility

Parameter	Description
`EXTRACT <group>`	EXTRACT <group> specifies the Extract process. You will create this process in GGSCI.

Table 39 Extract parameter file for the Reverse utility (continued)

Parameter	Description
`END {<time> | RUNTIME}`	<time> causes Extract to terminate when it reaches a record in the data source whose timestamp exceeds the one that is specified with this parameter. Valid values: ◆ <date> is a date in the format of yyyy-mm-dd. ◆ <time> is the time in the format of hh:mi[:ss[.cccccc]] based on a 24-hour clock. RUNTIME causes Extract to terminate when it reaches a record in the data source whose timestamp exceeds the current date and clock time. All unprocessed records with timestamps up to this point in time are processed. One advantage of using RUNTIME is that you do not have to alter the parameter file to change dates and times from run to run. Instead, you can control the process start time within your batch programming.

`[SOURCEDB <dsn>,]`
`[USERID <user id> [, PASSWORD <pw>`
`[<encryption options>]]]`
◆ SOURCEDB specifies a data source name, if required in the connection information. Not required for Oracle.
◆ USERID specifies database credentials and encryption information, if required. For Oracle, you can include a host string, for example:
`USERID ggs@ora1.ora, PASSWORD ...`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`NOCOMPRESSDELETES`	Causes Extract to send all column data to the output, instead of only the primary key. Enables deletes to be converted back to inserts.
`GETUPDATEBEFORES`	Directs Oracle GoldenGate to extract before images so that updates can be rolled back.
`RMTHOST <hostname>`	The name or IP address of the target system.
`{EXTTRAIL <input trail> |` `RMTTRAIL <input trail>}`	◆ Use EXTTRAIL to specify an extract trail on the local system. ◆ Use RMTTRAIL to specify a remote trail on a remote system. Specify the relative or full path name of a trail, including the two-character trail name, for example: `EXTTRAIL /home/ggs/dirdat/rt`

Table 39 Extract parameter file for the Reverse utility (continued)

Parameter	Description
`TABLE <owner.name>;`	The table or tables that are to be processed, specified with either multiple TABLE statements or a wildcard. Include any special selection and mapping criteria.

Example This example Extract parameter file uses a remote trail.

```
EXTRACT ext_1
END 2011-01-09 14:12:20
USERID ogg@ora1.ora, PASSWORD &
   AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
GETUPDATEBEFORES
NOCOMPRESSDELETES
RMTHOST sysb, MGRPORT 8040
RMTTRAIL /home/ggs/dirdat/in
TABLE tcustmer;
TABLE tcustord;
```

Table 40 Replicat parameter file for the Reverse utility

Parameter	Description
`REPLICAT <group>`	REPLICAT <group> specifies the Replicat process to be created in GGSCI.
`END {<time> | RUNTIME}`	<time> causes Replicat to terminate when it reaches a record in the data source whose timestamp exceeds the one that is specified with this parameter. Valid values: ◆ <date> is a date in the format of yyyy-mm-dd. ◆ <time> is the time in the format of hh:mi[:ss[.cccccc]] based on a 24-hour clock. RUNTIME causes Replicat to terminate when it reaches a record in the data source whose timestamp exceeds the current date and clock time. All unprocessed records with timestamps up to this point in time are processed. One advantage of using RUNTIME is that you do not have to alter the parameter file to change dates and times from run to run. Instead, you can control the process start time within your batch programming.

Table 40 Replicat parameter file for the Reverse utility (continued)

Parameter	Description
[TARGETDB <dsn>,] [USERID <user id> [, PASSWORD <password> [<encryption options>]]] ◆ TARGETDB specifies a data source name, if required in the connection information. Not required for Oracle. ◆ USERID specifies database credentials and encryption options, if required. For Oracle, you can include a host string, for example: USERID ggs@ora1.ora, PASSWORD ...	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.
{SOURCEDEFS <full_pathname>} | ASSUMETARGETDEFS ◆ Use SOURCEDEFS if the source and target tables have different definitions. Specify the source-definitions file generated by DEFGEN. See page 174 for more information about DEFGEN. ◆ Use ASSUMETARGETDEFS if the source and target tables have the same definitions.	Specifies how to interpret data definitions.
MAP <owner.name>, TARGET <owner.name>;	The table or tables (specified with either multiple MAP statements or a wildcard) to which to post the reversed data. When reversing data from the source database, the source and target TABLE entries are the same. When reversing replicated data from a target database, the source and target of each MAP statement are different.

Example The following is an example Replicat parameter file.

```
REPLICAT rep_1
END RUNTIME
USERID ogg@ora1.ora, PASSWORD &
   AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128 KEYNAME mykey1
ASSUMETARGETDEFS
MAP tcustmer, TARGET tcustmer;
```

Creating process groups and trails for reverse processing

To create process groups to perform the backout procedure, you create the following:

- An online Extract group.
- A local or remote trail that is linked to the Extract group. Extract captures the data from the database, and writes it to this trail, which serves as the input trail for the Reverse utility.
- An online Replicat group.

- Another local or remote trail that is linked to the Replicat group. This is the output trail that is written by the Reverse utility. Replicat reads this trail to apply the reversed data.

To create an Extract group for reverse processing

```
ADD EXTRACT <group name>, TRANLOG, BEGIN {NOW | <start point>}
```

Where:

- <group name> is the name of the Extract group. A group name can contain up to eight characters, and is not case-sensitive.
- TRANLOG specifies the transaction log as the data source.
- BEGIN specifies a starting timestamp at which to begin processing. 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.
  - <YYYY-MM-DD HH:MM[:SS[.CCCCCC]]> as the format for specifying an exact timestamp as the begin point.

To create an input trail that is linked to the Extract group

```
ADD {EXTTRAIL | RMTTRAIL} <pathname>, EXTRACT <group name>
[, MEGABYTES <n>]
```

Where:

- EXTTRAIL specifies a trail on the local system. RMTTRAIL specifies a trail on a remote system.
- <pathname> is the relative or fully qualified name of the input trail, including a two-character name that can be any two alphanumeric characters, for example c:\ggs\dirdat\rt. It must be the same name that you specified in the Extract parameter file.
- EXTRACT <group name> specifies the name of the Extract group.
- MEGABYTES <n> is an optional argument with which you can set the size, in megabytes, of each trail file (default is 100).

To create a Replicat group for reverse processing

```
ADD REPLICAT <group name>, EXTTRAIL <pathname>
[, BEGIN <start point> | , EXTSEQNO <seqno>, EXTRBA <rba>]
[, CHECKPOINTTABLE <owner.table>]
[, NODBCHECKPOINT]
```

Where:

- <group name> is the name of the Replicat group. A group name can contain up to eight characters, and is not case-sensitive.
- EXTTRAIL <pathname> is the relative or fully qualified name of an output trail that you will be creating for this Replicat with the ADD RMTTRAIL command.
- 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 records that are timestamped at the point when the ADD REPLICAT command is executed to create the group.

- `<YYYY-MM-DD HH:MM[:SS[.CCCCCC]]>` as the format for specifying an exact timestamp as the begin point.

- EXTSEQNO <seqno> specifies the sequence number of a file in the trail in which to start processing. EXTRBA <relative byte address> specifies a relative byte address as the start point 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 a trail file is c:\ggs\dirdat\aa000026, you would specify EXTSEQNO 26. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.
- CHECKPOINTTABLE <owner.table> specifies the owner and name of a checkpoint table other than the default specified in the GLOBALS file. To use this option, you must add the checkpoint table to the database with the ADD CHECKPOINTTABLE command (see "Creating a checkpoint table" on page 185).
- NODBCHECKPOINT specifies that this Replicat group will not use a checkpoint table.

To create an output trail that is linked to the Replicat group

```
ADD {EXTTRAIL | RMTTRAIL} <pathname>, REPLICAT <group name>
[, MEGABYTES <n>]
```

Where:

- EXTTRAIL specifies a trail on the local system. RMTTRAIL specifies a trail on a remote system.
- <pathname> is the relative or fully qualified name of the output trail, including a two-character name that can be any two alphanumeric characters, for example c:\ggs\dirdat\rt. This must be the trail that was specified with EXTTRAIL in the ADD REPLICAT command.
- REPLICAT <group name> specifies the name of the Replicat group.
- MEGABYTES <n> is an optional argument with which you can set the size, in megabytes, of each trail file (default is 100).

Running the Reverse utility

1. From GGSCI, run Extract.

   ```
   START EXTRACT <group>
   ```

2. Issue the following command until "at EOF" is returned, indicating that Extract is finished capturing the specified records.

   ```
   SEND EXTRACT <group>, STATUS
   ```

3. Run the Reverse utility by using the fully qualified path name or by changing directories to the Oracle GoldenGate directory and running reverse from there.

   NOTE Using a full path name or a path relative to the Oracle GoldenGate directory is especially important on UNIX systems, to prevent confusion with the UNIX reverse command.

   ```
   /<GoldenGate_directory>/reverse <input file>, <output file>
   ```

Where:

- <input file> is the input file specified with EXTTRAIL or RMTTRAIL in the Extract parameter file.
- <output file> is the output file specified with EXTTRAIL in the ADD REPLICAT command.

Example

```
\home\ggs\reverse input.c:\ggs\dirdat\et, output.c:\ggs\dirdat\rt
```

4. When reverse is finished running, run Replicat to apply the reversed-out data to the database.

```
START REPLICAT <group>
```

Undoing the changes made by the Reverse utility

If the reverse processing produces unexpected or undesired results, you can reapply the original changes to the database. To do so, edit the Replicat parameter file and specify the *input* file in place of the output file, then run Replicat again.

APPENDIX 1
Supported character sets

Oracle GoldenGate supports the following character sets. The identifiers that are shown should be used for Oracle GoldenGate parameters or commands when a character set must be specified.

Identifier	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
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

Identifier	Character set
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
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

Identifier	Character set
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
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 EBCIDC, 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

Identifier	Character set
IBM01146	IBM 1146-1/695-1 EBCDIC, United Kingdom
IBM290	IBM 290 EBCDIC, Japan (Katakana) Extended
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)
ibm-1097	IBM 1097 EBCDIC, Farsi
ibm-1018	IBM 1018 EBCDIC, Finland Sweden (ISO-7)

Identifier	Character set
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, Ukranian 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	Ukranian (KOI8-U)
eucTH	EUC Thai
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

Identifier	Character set
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
eucjis	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

Identifier	Character set
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
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

APPENDIX 2
Supported locales

Oracle GoldenGate supports the following locales. Oracle GoldenGate uses the locale 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		

APPENDIX 3
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.

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 (unless the NOHEADERS parameter was specified), 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.

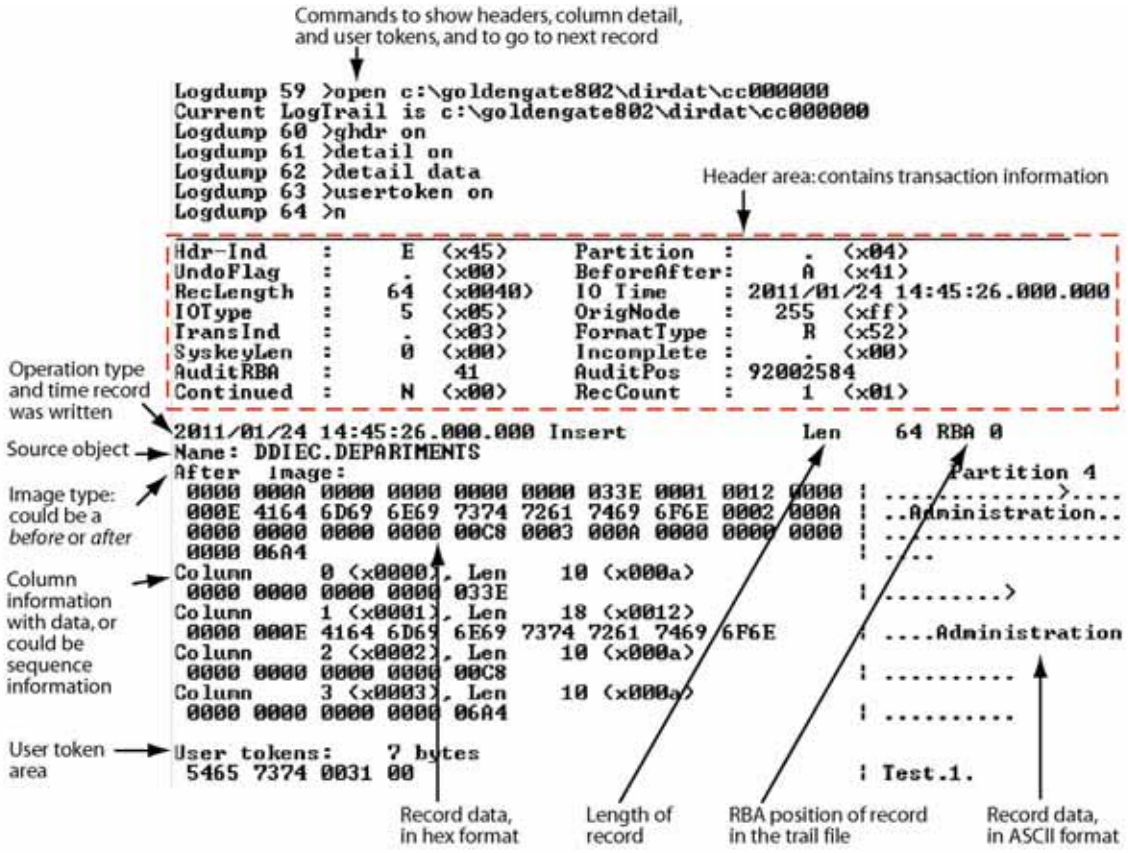
Oracle GoldenGate trail files are unstructured. You can view Oracle GoldenGate records with the Logdump utility provided with the Oracle GoldenGate software. For more information, see the Logdump documentation in the *Windows and UNIX Troubleshooting and Tuning Guide*.

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.

Figure 26 Sample trail record as viewed with the Logdump utility



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

The following describes the fields of the Oracle GoldenGate record header. Some fields apply only to certain platforms.

Table 41 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 42 on page 302 for a list of operation types.
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	The relative byte address of the commit record. All records in a transaction will have the same commit relative byte address. The combination of IO Time and AuditRBA uniquely identifies data from a given transaction.

Table 41 Oracle GoldenGate record header fields (continued)

Field	Description
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. 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	This field is for Oracle GoldenGate internal use and may not be meaningful for any particular database. 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: ◆ 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. Inserts are always after images, deletes are always before images.
IO Time	The timestamp of the commit record, in local time of the source system, in GMT format. All records in a transaction will have the same commit timestamp. The combination of IO Time and AuditRBA uniquely identifies data from a given transaction.
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.

Table 41 Oracle GoldenGate record header fields (continued)

Field	Description
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 of the Extract process in the transaction log.
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.

- GGS_TRANS_TIMESTAMP
- GGS_TRANS_RBA
- GGS_OP_TYPE
- GGS_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. The terms “full” and “compressed” image format are used in the descriptions. These terms are used in a different context here than when they are used in other parts of the documentation in reference to how Extract writes column data to the trail, meaning whether only the key and changed columns are written (“compressed”) versus whether all columns are written to the trail (“uncompressed” or “full image”).

Full record image format

Full record image format is only 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 format

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 data> 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:

```
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 42 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 COMPRESSDELETES 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 42 Oracle GoldenGate operation types (continued)

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 "15-FieldComp") 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 offset><2-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 offset><2-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

Table 42 Oracle GoldenGate operation types (continued)

Type	Description	Platform
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
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
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.	NSK non-TMF

Table 42 Oracle GoldenGate operation types (continued)

Type	Description	Platform
	◆ 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.	
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
249 through 254	These are different types of NonStop trace records. Trace records are used by Oracle GoldenGate support analysts. The following are descriptions. ◆ 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. Depracated 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 the Logdump documentation in the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.

APPENDIX 4
About the commit sequence number

When working with Oracle GoldenGate, you might need to refer to a *Commit Sequence Number*, or CSN. The CSN can be required to position Extract in the transaction log, to reposition Replicat in the trail, or for other purposes. It is returned by some conversion functions and is included in reports and certain GGSCI output.

A CSN is an identifier that Oracle GoldenGate constructs to identify a transaction for the purpose of maintaining transactional consistency and data integrity. It uniquely identifies a point in time in which a transaction commits to the database.

Each kind of database management system generates some kind of unique serial number of its own at the completion of each transaction, which uniquely identifies that transaction. A CSN captures this same identifying information and represents it internally as a series of bytes, but the CSN is processed in a platform-independent manner. A comparison of any two CSN numbers, each of which is bound to a transaction-commit record in the same log stream, reliably indicates the order in which the two transactions completed.

The CSN value is stored as a token in any trail record that identifies the beginning of a transaction. This value can be retrieved with the @GETENV column conversion function and viewed with the Logdump utility.

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, such as the Sybase CSN.

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 43 Oracle GoldenGate CSN values per database

Database	CSN Value
DB2 for i	There is no CSN for DB2 for i, because extraction (capture) is not supported by Oracle GoldenGate for this database.

DB2 LUW	`<LSN>`
	Where:
	◆ <LSN> is the variable-length, decimal-based DB2 log sequence number.
	Example:
	`1234567890`

Table 43 Oracle GoldenGate CSN values per database (continued)

Database	CSN Value

DB2 z/OS	`<RBA>`
	Where:
	◆ <RBA> is the 6-byte relative byte address of the commit record within the transaction log.
	Example:
	`1274565892`

MySQL	`<LogNum>:<LogPosition>`
	Where:
	◆ <LogNum> is the the name of the log file that contains the START TRANSACTION record for the transaction that is being identified.
	◆ <LogPosition> is the event offset value of that record. Event offset values are stored in the record header section of a log record.
	For example, if the log number is 12 and the log position is 121, the CSN is:
	`000012:000000000000121`

Oracle	`<system change number>`
	Where:
	◆ <system change number> is the Oracle SCN value.
	Example:
	`6488359`

SQL/MX	`<sequence number>.<RBA>`
	Where:
	◆ <sequence number> is the 6-digit decimal NonStop TMF audit trail sequence number padded with leading zeroes.
	◆ <RBA> is the 10-digit decimal relative byte address within that file, padded with leading zeroes.
	Together these specify the location in the TMF Master Audit Trail (MAT).
	Example:
	`000042.0000068242`

Table 43 Oracle GoldenGate CSN values per database (continued)

Database	CSN Value

SQL Server	Can be any of these, depending on how the database returns it:
	◆ 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
	Where:
	◆ 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.
	Examples:
	`0X00000d7e:0000036b:01bd`
	`0000003454:0000000875:00445`
	`0Xd7e:36b:1bd`
	`3454:875:445`
	`3454000000087500445`

Sybase	`<time_high>.<time_low>.<page>.<row>`
	Where:
	◆ <time_high> and <time_low> represent an instance ID for the log page. It is stored in the header of each database log page. <time_high> is 2-bytes and <time_low> is 4-bytes, each padded with leading zeroes.
	◆ <page> is the database logical page number, padded with zeroes.
	◆ <row> is the row number, padded with zeroes.
	Taken together, these components represent a unique location in the log stream. The valid range of a 2-byte integer for a timestamp-high is 0 - 65535. For a 4-byte integer for a timestamp-low, it is: 0 - 4294967295.
	Example:
	`00001.0000067330.0000013478.00026`

Teradata	`<sequence ID>`
	Where:
	◆ <sequence ID> is a generic fixed-length printable sequence ID.
	Example:
	`0x08000000000000000D700000021`

TimesTen	There is no CSN for TimesTen, because extraction (capture) is not supported by Oracle GoldenGate for this database.

Glossary

The following explains terminology contained in this manual.

Term	Definition
abend	*Abnormal end*. The failure or unexpected termination of a process running on a computer system.
after image	The values of a row in a database after an insert or update is performed.
alias Extract	An Extract group that operates on a target system that resides within a more secure network zone than the source system. The purpose of the alias Extract is to initiate TCP/IP connections from the target to the less-trusted source. Once a connection is established, data is processed and transferred across the network in the usual manner by a passive Extract group that operates on the source system.
append mode	The default method of writing to the trail, whereby Extract appends re-read data to the trail file after a failure, instead of overwriting the old data.
Archived Log Only mode (ALO)	A mode of operation for Extract, where the process is configured to read exclusively from the archived transaction logs on a production or standby database system.
batch Replicat processing mode	In batch mode, Replicat organizes similar SQL statements into arrays and then applies them at an accelerated rate. Replicat batches the statements within a memory queue and then applies each batch in one database operation. The behavior of this mode is controlled by the BATCHSQL parameter. See also normal Replicat processing mode.
audit trail	A file on a NonStop Server system that stores modifications made to a database for the purpose of replication and recovery.
before image	The values that exist in a row in a database before a SQL operation is performed on that row.
bidirectional synchronization	Permits load distribution across multiple databases and servers where, in most cases, different users can change the same sets of data and those changes are synchronized by Oracle GoldenGate.
BLOB	See *LOB*.

Term	Definition
Bounded Recovery	Part of the Extract recovery system. Bounded Recovery guarantees an efficient recovery if Extract stops in an unplanned manner and then is started again, no matter how many open transactions there were at the time that Extract stopped, nor how old they were. It sets an upper boundary for the maximum amount of time that it would take for Extract to recover to the point where it stopped and then resume normal processing.
caller	The Oracle GoldenGate process that executes a user exit routine.
canonical format	A data format that Oracle GoldenGate uses to store data in a trail or extract file. This format allows data to be exchanged rapidly and accurately among heterogeneous databases.
cascading synchronization	An Oracle GoldenGate configuration in which data is sent from a source system to one or more intermediary systems and, from those systems, to one or more other systems in a synchronized state.
change synchronization	The process of synchronizing data changes made to a database on one system with a similar set of data on one or more other systems.
checkpoint file	A file on disk that stores the checkpoint generated by Oracle GoldenGate processes.
checkpoint table	A table created in the target database that maintains Replicat checkpoints, used optionally in conjunction with a standard checkpoint file on disk.
checkpoints	Internal indicators that record the current read and write position of an Oracle GoldenGate process. Checkpoints are used by the Extract and Replicat processes for online change synchronization to ensure data accuracy and fault tolerance.
CLOB	See *LOB*.
CMDSEC file	An Oracle GoldenGate file that stores rules for GGSCI command permissions.
Collector	The process that receives data from the Extract process over TCP/IP and writes it to a trail or extract file on the target system.
collisions	Errors that occur when data changes that are replicated by Oracle GoldenGate are applied to a target table, but the target row is either missing or is a duplicate.
column	One among a set of attributes assigned to an entity that is described by a database table. For example, there can be columns for the name, address, and phone number of the entity called “employees.”

Term	Definition
column map	See *map*.
column-conversion functions	Built-in Oracle GoldenGate processing functions that perform comparisons, tests, calculations, and other processing for the purpose of selecting and manipulating data.
commit	A transaction-control statement that ends a transaction and makes permanent the changes that are performed by the SQL statements within that transaction.
Commit Sequence Number (CSN)	A CSN is an identifier that Oracle GoldenGate constructs to identify a transaction for the purpose of maintaining transactional consistency and data integrity. It uniquely identifies a particular point in time in which a transaction commits to the database. The composition and value of the CSN varies, depending on the type of database that generated the transaction. A CSN captures the unique information that a database uses to identify transactions and represents it internally as a series of bytes, but Oracle GoldenGate processes the CSN in a platform-independent manner.
compressed update	A method of logging SQL update operations by which only column values that changed as the result of the update are logged to the transaction log.
conflict resolution	Instructions used in bidirectional synchronization that provide processing and error-handling rules in the event that the same SQL operation is applied to the same row in two or more databases at (or about) the same time.
consolidated synchronization	The process of replicating different data from two or more databases to one central database, such as in data warehousing.
conversion	See *transformation*.
data definitions file	See *source definitions file* and *target definitions file*.
data pump	A secondary Extract process that reads from an extract file or trail. The trail is populated by a primary Extract process that reads from the data source.
data source	The container of the data changes that are to be processed by Oracle GoldenGate. A data source can be: ◆ the transaction log of a database ◆ a Vendor Access Module

Term	Definition
data source name (DSN)	A DSN defines an ODBC connection to a database. A DSN consists of a database name, the database directory, the database ODBC driver name, database authentication information, and other information depending on the database. External applications, such as Oracle GoldenGate, require a DSN, because a DSN enables an application to connect to a database without having to encode the required information within the application program. The three types of DSN are ◆ A system DSN can be used by any entity that has access to the machine. It is stored within the system configuration. ◆ A user DSN can only be used by a specific user. It is stored within the system configuration. ◆ A file DSN is stored in a text file with a .dsn extension. It can be shared among different systems where the required ODBC driver is installed.
data type	An attribute of a piece of data that identifies what kind of data it is and what kinds of operations can be performed on it. For example, an integer data type is a number, and a character data type contains letters.
DDL	*Data Definition Language*. Data that defines the structure of a database, including rows, columns, tables, indexes, and database specifics such as file locations, users, privileges, and storage parameters.
DEFGEN	An Oracle GoldenGate utility that generates a data definitions file.
discard file	An Oracle GoldenGate file containing information about SQL operations that failed. This file is created when a record cannot be processed, but only if the DISCARDFILE parameter exists in the parameter file to specify the location for the file.
DML	*Data Manipulation Language*. Retrieves and manipulates data in a database. In the case of SQL, the actions are “select”, “insert”, “update”, and “delete”.
DSN	See data source name (DSN).
dynamic Collector	A Collector process that the Manager process starts automatically, as opposed to a static Collector.
EMSCLNT	An Oracle GoldenGate utility that distributes Oracle GoldenGate system error messages that originate on Windows and other supported operating systems to the EMS (Event Management Subsystem) server on the NonStop Server.
ENCKEYS file	An Oracle GoldenGate lookup file that stores encryption keys.

Term	Definition
encryption	A method of encoding data into a format that is unreadable to anyone except those who posses a password or decryption code to decipher it.
error log	A file that shows processing events, messages, errors, and warnings generated by Oracle GoldenGate. Its name is ggserr.log and it is located in the root Oracle GoldenGate directory.
event marker system	A system that customizes Oracle GoldenGate to take a specific action during processing based on a record that qualifies for filtering criteria. For example, you can skip the record or stop the Oracle GoldenGate process when the record is encountered. See also event record.
event record	A record in the transaction log that satisfies specific filter criteria and is used to trigger a specific action during processing. See also event marker system.
exceptions map	A special MAP parameter used specifically for error handling, which executes only after an error and sends error data to an exceptions table.
exceptions table	A database table to which information about failed SQL operations is written as the result of an exceptions map. Used for error handling.
Extract	The Oracle GoldenGate program that reads data either from a data source, from source tables, or from a local trail or file. Extract processes the data for delivery to the target system. A *primary Extract* reads the data source or database tables, and a *data-pump Extract* reads a local trail that is populated by a primary Extract.
extract file	A file written by Oracle GoldenGate where data is stored temporarily awaiting further processing during an initial load.
extraction	The processing of reading data from database tables or from a data source in preparation for further processing and/or transmission to a target database.
fetch	A query to the database issued by the Extract process when processing a record from the transaction log. A fetch is required if the data values that are needed to complete the SQL operation are not present in the record.
file header	See header.
filtering	The use of rules to select and exclude data for extraction or replication.
function	A segment of code that can be executed within an application or routine. See also *column-conversion functions*.

Term	Definition
GGSCI	*GoldenGate Software Command Interface*. The primary interface for issuing commands that configure, control, and monitor Oracle GoldenGate.
GLOBALS file	A text file in the root Oracle GoldenGate directory that contains parameters which apply to the Oracle GoldenGate instance as a whole, as opposed to runtime parameters that are specific to a process such as Extract or Replicat.
group	Also known as *process group*. A group consists of an Oracle GoldenGate process (either Extract or Replicat) and the parameter file, the checkpoint file, and any other files associated with that process.
header	A header can be: ◆ **A record header**: an area at the beginning of a record in an Oracle GoldenGate trail file that contains information about the transaction environment for that record. ◆ **A file header**: an area at the beginning of each file in a trail, or at the beginning of an extract file. This header contains information about the file itself, such as the Oracle GoldenGate version.
heterogeneous	A data environment where data is being exchanged among different types of applications, different types of databases, or different operating systems, or among a combination of those things.
homogeneous	A data environment where data is being exchanged among identical types of applications, databases, and operating systems.
initial load	The duplication of source data into a target database to make the two databases identical.
intermediary system	A system on the network that serves as a transfer station between the source and target systems. This system can be host to additional processing activities, such as transformation.
key	A column or columns in a table that are being used as a unique identifier for the rows in that table. Oracle GoldenGate uses the key to find the correct row in the target database and for fetches from the source database. For Oracle GoldenGate, a key can be the primary key, a unique key, a substitute key, or all of the columns of a table in the absence of a defined identifier.
KEYCOLS	A clause in a TABLE or MAP statement that defines a column or columns for Oracle GoldenGate to use as a unique identifier to locate any given row in a table.
KEYGEN	An Oracle GoldenGate utility that generates encryption keys.

Term	Definition
lag	Extract lag is the difference between the time that a record was processed by Extract and the timestamp of that record in the data source. Replicat lag is the difference between the time that the last record in a trail was processed by Replicat and the timestamp of the record in the trail.
latency	The difference in time between when a change is made to source data and when that change is reflected in the target data.
LOB	*Large Object*. A data type in a database that represents an unstructured object that is too large to fit into a character field, such as a Microsoft Word document or a video or sound file. Subsets of LOB are CLOB (Character Large Object) and BLOB (Binary Large Object), which contain character data and binary data, respectively.
log-based extraction	A method of extracting data changes from the database transaction log.
logical name	A name for a stored procedure that represents an instance of the *execution* of the procedure, as opposed to its actual name. For example, logical names for a procedure named “lookup” might be “lookup1,” “lookup2,” and so forth.
LUW	*Linux, UNIX, Windows*. An acronym that describes an application that runs on any of these platforms, such as DB2 LUW.
macro	A computer program that automates a task, such as the implementation of parameters and commands.
Manager	The control program for Oracle GoldenGate processing.
map	An association between a set of source data and a set of target data. A map can include data selection and conversion criteria. These maps are specified in a Replicat MAP parameter.
MAP statement	A Replicat parameter that specifies the relationship between a source table and a target table and the processing rules for those tables.
marker	A record that is inserted into the audit trail on a NonStop Server to identify application-specific events in the context of Extract and Replicat processing. See also event marker system.

Term	Definition
normal Replicat processing mode	The default processing mode for Replicat. In its normal mode, Replicat accumulates operations from multiple source transactions, in transaction order, and applies them as a group within one transaction on the target to improve performance. The GROUPTRANSOPS parameter controls the number of operations that are in this transaction, but the boundary can be adjusted automatically by Replicat to ensure that all operations from the last transaction in the group are included. See also batch Replicat processing mode and source Replicat processing mode.
object	For the purpose of this documentation, the term *object* refers to any logical component of a database that is visible to, and can be created by, its users for the purpose of storing data (for example, tables), defining ownership and permissions (for example, roles), executing an action on another object (for example, triggers), and so forth.
object record	A file containing attributes of the tables and other database objects that are configured for processing by Oracle GoldenGate, such as column IDs and data types.
ODBC	*Open Database Connectivity*. Acronym for a standard interface that enables applications to connect to different types of databases in a uniform manner. The goal of ODBC is to make the process of connecting to a database independent of programming languages, database systems, and operating systems.
online change synchronization	An Oracle GoldenGate processing method in which Extract and Replicat processes run continuously to synchronize data changes unless they are stopped by an Oracle GoldenGate user. Online processes maintain checkpoints in the trail.
online Extract	An Extract group that is configured for online change synchronization.
online processing	See *online change synchronization*.
online Replicat	A Replicat group that is configured for online change synchronization.
operation	A single unit of work. This typically refers to a SQL change made to data or a change made to the structure of an object in the database, but can also refer to any work done by a computer process.

Term	Definition
Oracle GoldenGate Director	Graphical user interface software that enables Oracle GoldenGate users to monitor and manage Oracle GoldenGate processes. The components of Oracle GoldenGate Director are: **Oracle GoldenGate Director Administrator**: A utility used by administrators to define users and instances of Oracle GoldenGate. **Oracle GoldenGate Director Server**: A software module that gathers data about the Oracle GoldenGate processes. **Oracle GoldenGate Director Client**: Software installed on a user's system as an interface to Oracle GoldenGate Director. **Oracle GoldenGate Director Web**: A browser-based user interface to Oracle GoldenGate Director (requires no software to be installed).
Oracle GoldenGate Rollback	A utility that uses before images to undo changes made to a database.
overwrite mode	A method of writing data to the trail that was used in Oracle GoldenGate versions prior to version 10.0. In this mode, Extract overwrites existing data upon recovery, instead of appending it to the end of the trail file.
owner	A logical namespace in a database to which database objects are assigned as part of the organizational hierarchy. Because the ownership of database objects is managed differently by different database types, the term *owner* is used in this documentation to denote whichever entity is recognized by the database as the qualifier of an object name, typically a user or schema name. For example, in a qualified Oracle table name of scott.emp, the owner is scott.
parameter	An input or output value for a computer program, such as the code of an application like Oracle GoldenGate, a stored procedure, a macro, script, or other processing instructions.
parameter file	A file containing parameters that control the behavior of an Oracle GoldenGate process. The default location for parameter files is the dirprm directory in the Oracle GoldenGate installation directory.
pass-through data pump	A data pump that is configured with the PASSTHRU parameter to bypass the need to look up data definitions. This enables faster processing and enables a pump to be used on an intermediary system that has no database.
pass-through Extract	See *pass-through data pump*.

Term	Definition
passive Extract	An Extract process that operates on the source system when an alias Extract is being used on the target. This Oracle GoldenGate configuration is required when security rules do not permit TCP/IP connections to be initiated from the source system (as a typical Extract would do) because the target is inside a more secure network zone. The passive Extract is the data pump, when one is being used; otherwise, it is the primary Extract.
primary Extract	An Extract group that reads from the data source or directly from the database tables. A primary Extract can write to a local trail, which is then read by a data pump Extract, or it can send the data across TCP/IP to the target system.
primary key	An integrity constraint consisting of a column or columns that uniquely identify all possible rows that exist in a table, current and future. There can be only one primary key for a table. A primary key contains an implicit NOT NULL constraint.
process report	A report generated for Extract, Replicat, and Manager that provides information about the process configuration and runtime statistics and events. The default location for process reports is the dirrpt directory of the Oracle GoldenGate installation directory.
record	A unit of information in a transaction log or trail that contains information about a single SQL operation performed on a row in a database. The term *record* is also used to describe the information contained in a specific row of a table.
record header	See header.
remote file	An extract file on a remote system.
remote trail	A trail on a remote system.
Replicat	The Oracle GoldenGate process that applies data to target tables or moves it to another application or destination.
replication	The process of recreating source database operations and applying them to a target database.
report	See *process report*.
report file	See *process report*.
rollback	The act of undoing changes to data that were performed by SQL statement within an uncommitted transaction.
rollover	The closing of one file in a sequence of files, such as a trail, and the opening of a new file in the sequence.

Term	Definition
routine	A segment of code that is executed within an application such as Oracle GoldenGate, which calls functions that retrieve and return values and provide responses. See also *user exit*.
row	Information about a single instance of an entity, such as an employee, that is stored within a database table. For example, a row stores information about "John Doe" in relation to the broader collection of rows that stores information about John and the other employees in a company. Also commonly known as a *record*.
source	The location of the original data that Oracle GoldenGate will be extracting, as in *source database* and *source system*.
source definitions file	A file containing the definitions of the source tables, which is transferred to the target system. This file is used by the Replicat process for data conversion when the source and target tables are dissimilar.
source Replicat processing mode	In source processing mode, Replicat applies SQL operations within the same transaction boundaries that were used on the source. See also normal Replicat processing mode.
statement	An elementary instruction in a computer programming language, for example a SQL statement, parameter statement, or command statement.
static Collector	A Collector process that is started manually by an Oracle GoldenGate user, instead of being started automatically by the Manager process.
stored procedure	A group of SQL, PL/SQL, or Java statements that are stored in the database and called on demand by a process or application to enforce business rules, supplement application logic, or perform other work as needed.
substitute key	A unique identifier that consists of any columns in a table that can uniquely identify the rows in that table. A substitute key is not defined in the definition of a table; it is created by creating a KEYCOLS clause in a TABLE or MAP statement.
synchronization	The process of making or keeping two or more sets of data consistent with one another. To be consistent, one set might be identical to the other, or one set might be a reorganized, reformatted, or expanded version of the other, while retaining the essence of the information itself.
table	A logical unit of storage in a database that consists of rows and columns, which together identify the instances of a particular entity (for example, "employees") and the attributes of that entity, such as name, address, and so forth.

Term	Definition
TABLE statement	An Extract parameter that specifies a source table or tables whose data is to be extracted from the database.
TAM (Teradata Access Module)	An interface between the Change Data Capture (CDC) component of a Teradata database and the Extract process. It allows Oracle GoldenGate to communicate with the Teradata replication components.
target	The destination for the data that is processed by Oracle GoldenGate, as in *target database* and *target system*.
target definitions file	A file containing the definitions of the target tables. This file is transferred to the source system and is used by the Extract process for data conversion when the source and target tables are dissimilar.
task	A special type of initial load in which the Extract process communicates directly with the Replicat process over TCP/IP instead of using a Collector process or trail.
token	A user-defined piece of information that is stored in the header portion of a record in the Oracle GoldenGate trail file. Token data can be used to customize the way that Oracle GoldenGate delivers information.
trace table	A special table created for use by Oracle GoldenGate in an Oracle database. The table is used in conjunction with parameter settings to prevent replicated data from being sent back to the source in a bidirectional synchronization configuration.
trail	A series of files on disk where Oracle GoldenGate stores data temporarily in preparation for further processing. Oracle GoldenGate records checkpoints in the trail for online change synchronization.
transaction	A group of one or more SQL operations (or statements) that are executed as a logical unit of work within a set of beginning and ending transaction-control statements. As a unit, all of the SQL statements in a transaction must execute successfully, or none of the statements can execute. A transaction is part of a system of database measures that enforce data and structural integrity.
transaction log	A set of files that records all of the SQL change operations performed on a database for the purpose of data recovery or replication.
transformation	Also called *conversion*. The process of manipulating source data to the format required by target tables or applications, for example converting dates or performing arithmetic calculations. You can do transformation by means of the Oracle GoldenGate column-conversion functions.

Term	Definition
unidirectional synchronization	A configuration where data changes are replicated in one direction, source-to-target. Changes cannot be made to that same data and then sent back to the source, as is the case in a bidirectional configuration.
unique key	An integrity constraint consisting of a column or columns that uniquely identify all possible rows that exist in a table, current and future. Differs from a primary key in that it does not have an implicit NOT NULL constraint. There can be more than one unique key on a table.
Unit of Work	A set of data operations that are executed as a logical unit in a database, where all must succeed or none can succeed. In IBM terminology, the term *unit of work* is the equivalent of the term transaction in other types of databases.
user exit	A user-created program written in C programming code that is called during Oracle GoldenGate processing to perform custom processing such as to convert data, to respond to database events, and to repair invalid data.
VAM (Vendor Access Module)	An API interface that is used by an Oracle GoldenGate process module to communicate with certain kinds of databases.
VAM trail	A series of files, similar to a transaction log, that are created automatically and aged as needed. Data operations from concurrent transactions are recorded in time sequence, as they occur, but not necessarily in transaction order. Used to support the Teradata maximum protection commit protocol.
wildcard	A placeholder for an unknown or unspecified character or set of characters. A wildcard is a means of specifying multiple names in a parameter or command statement. Oracle GoldenGate supports the asterisk (*) wildcard, which represents any number of unknown characters.

Index

Symbols

@ABSENT 157
@CASE function 165
@COLSTAT function 164
@COLTEST function 164
@COMPUTE function 154, 162
@DATENOW function 171
@EVAL function 165
@GETENV function 159, 166, 171
@IF function 164
@NULL function 157
@NUMBIN function 163
@PRESENT 157
@STR* functions 163
@TOKEN function 166
@VALONEOF function 165
*** wildcard character** 32, 37

Numerics

256-key byte substitution 125

A

ABEND option
- REPERROR 169
- TCP errors 140, 173

action, triggering during processing 242
active-active configuration, creating 93
ADD EXTRACT command 187, 189, 282
ADD EXTTRAIL command 190, 282, 283
ADD REPLICAT command 194, 282
ADD RMTTRAIL command 190, 282, 283
adding
- checkpoint table 185
- Extract group 187, 189, 262, 282
- objects to extraction 269
- parameters 31
- Replicat group 194, 266, 282
- trail 190
- see also *creating*

Advanced Encryption Security (AES) 125
alias Extract 138
ALLOWNESTED command, GGSCI 24
ALTER EXTTRAIL command 275
ALTER RMTTRAIL command 275
append recovery mode 14
architecture, Oracle GoldenGate 10
archive logs, purging 275
arithmetic operations
- during transformation 162
- in user exits 240
- in WHERE clause 156
- with before values 158

ASSUMETARGETDEFS parameter 147
asterisk wildcard character 32, 37
AUTORESTART parameter 20
AUTOSTART parameter 20
AUTOSTART parameters 197

B

balances, calculating 158
batch scripts 24
before values, using 157, 277
BEGIN argument, ADD EXTRACT 188, 194, 282
BEGIN macro keyword 234
bidirectional configuration, creating 93
Blowfish encryption 125, 126
bulk data load 219

BULKLOAD parameter 222
byte substitution encryption 125

C

calculations, arithmetic 158
callback routines in user exits 241
canonical format, of trail data 14
cascading synchronization, configuring 55
case sensitivity
CMDSEC name 136
group name 187, 194
in macro statements 234
in parameter declaration 33
token name 166
C-code macros, using 239
centralized reporting 70
change sequence number
about 307
changing
data structure 160
database objects 269
file format 295
macro character 234
Oracle GoldenGate process configuration 261
parameters 31
TCP/IP error handling 172
text editor 28
trail files, size of 275
character set preservation 41
character set support 41
character, macro 234
characters
comparing 157
manipulating 163
matching with wildcard 37
supported in object names 33
CHARSET parameter 25
CHECKPARAMS parameter 30
checkpoint table
specifying to Extract 97
using 185
checkpoints
about 14
initial, creating 188, 194, 282
CHECKPOINTTABLE option, ADD REPLICAT 194, 283
circular replication 96
cluster, running Manager on 21
CMDSEC file 125, 136
CMDTRACE parameter 239
Collector, about 15
COLMAP option, TABLE or MAP 146, 149, 226
COLMATCH parameter 149
COLS and COLSEXCEPT options, TABLE 158
COLSTAT function 164
COLTEST function 164
column-conversion functions 146, 153
columns
adding to table 269
availability, ensuring 156
fetching for filters 157, 231
mapping 146, 236
null or missing 164
selecting and excluding 158
testing and transforming 157, 160
commands
authorization for 136
automating 24
database 226
GGSCI 23
comments in
parameter file 29
Commite Sequence Number (CSN), about 17
comparing
before and after values 157
column values 157
COMPUTE function 154, 162

configuring
active-active (bi-directional) 93
change data synchronization 184
data distribution (one to many) 63
data warehousing (many to one) 70
initial data load 200
live standby 77
Manager 20
reporting, cascading 55
reporting, standard 45
reporting, with data pump on intermediary system 50
reporting, with source data pump 47
security
data 126, 133
GGSCI commands 136
password 130
source definitions 178
conflict resolution 105
connections, network, see *network*
consolidated synchronization, planning for 70
continuous change synchronization 184
controlling Oracle GoldenGate 23, 25, 239
conversion functions, Oracle GoldenGate 153, 165
converting data 142
copy utility, for initial load 203
creating
encryption keys 134
initial checkpoint 188, 194, 282
parameter files 28
source-definitions file 175
trail 190
user exits 240
see also *adding*
CSN, for supported databases 17
CSN, see *change sequence number*
CUSEREXIT parameter 241
custom programming, using 226

D

data
encrypting 125
extracting, see *extracting data*
filtering 153
loading 200
looping, preventing 100, 102
mapping and manipulating 142, 226, 239
replicating, see *replicating data*
storing 12
data distribution configuration, creating 63
data looping, preventing 96
data pump
adding 188, 264
multi-target configuration 63
overview of 12
pass-through mode 12
data source, description 11
data types
converting 160
mapping 152
data warehousing configuration, creating 70
database
attributes, changing 269
command, executing from Oracle GoldenGate 226
password, encrypting 130
procedures and queries, using 226
types supported 10
data-definitions file, creating 174
dates
mapping 152
transforming 162
DB2
bidirectional synchronization 97
bootstrap data set, specifying 187
supported processing methods 10
DB2 for i, supported processing methods 10
DBOP option, SQLEXEC 232
DECRYPTTRAIL parameter 127
DEFAULT option
ENCRYPTKEY 131, 132
REPERROR 168

DEFAULTUSERPASSWORD option, DDLOPTIONS 132
DEFERAPPLYINTERVAL parameter 196
DEFGEN 175
definitions template, using 176
definitions, generating 175
DEFSFILE parameter 178
delaying
  Replicat transactions 196
DELETE EXTRACT command 268, 274
deletes, converting
  to inserts during reverse processing 277
  to inserts or updates 159
DESC option, ADD EXTRACT 188, 189
direct bulk load to SQL*Loader 219
direct load, Oracle GoldenGate 214
discard file 255
DISCARD option
  EVENTACTIONS 243
  REPERROR 169
DISCARDFILE parameter 256
DISCARDROLLOVER parameter 256
DSOPTIONS parameter 193
dual-stack IPv6 19
DYNAMICPORTLIST parameter 19, 215

E

EDIT PARAMS command 28
editing
  CMDSEC file 125, 136
  ENCKEYS file 135
  parameter file 31
  see also *changing*
editor, changing 28
EMSCLNT 257
ENCKEYS file 135, 136
ENCRYPT option, RMTHOST 133
encrypting
  data 125
  password 130
encryption
  password in IDENTIFIED BY 132
ENCRYPTKEY option
  DDLOPTIONS 132
END macro keyword 234
environment
  information, capturing 166
  variables in parameter file 32
ER commands 198
ERCALLBACK function 241
ERROR option, SQLEXEC 231
errors
  during
    bidirectional synchronization 105
    stored procedures 231
  handling 168
  process 253
  response options 168
  SQL 255
  TCP/IP 172
event marker system 242
event record 242
Event Viewer, Oracle GoldenGate messages in 256
events
  handling 168
  monitoring 247
  triggering during processing 242
EXCEPTION option, REPERROR 169
exceptions handling, configuringt 169
EXCEPTIONSONLY in MAP statement 169
EXCLUDETRANS option, TRANLOGOPTIONS 98
EXCLUDEUSER option, TRANLOGOPTIONS 98
EXCLUDEUSERID option, TRANLOGOPTIONS 98, 100, 102
excluding
  columns 158
  Replicat transactions 96
  rows 153
  transactions from capture 100, 102
exit routines, using 239

Extract
about 11
alias 138, 139
data pump, using 12
errors, handling 168
group, adding
to active configuration 262
to new configuration 187
passive 138
running
from GGSCI 196
EXTRACT argument, ADD RMTTRAIL, ADD EXTTRAIL 190, 282
extract file, about 14
EXTRACT parameter 191
extract trail, see *trail*
extracting data
about 11
for initial load 200
from trail 12
EXTRBA option, ADD REPLICAT 194
EXTSEQNO option, ADD REPLICAT 194, 283
EXTTRAIL option, ADD REPLICAT 194, 282
EXTTRAILSOURCE option
ADD EXTRACT 188

F

failover configuration, creating 77
FastLoad, Teradata 224
FETCHBEFOREFILTER option, TABLE 157
FETCHCOLS options, TABLE 157, 231
fetching column values for filters 157, 231
FieldComp record 303
field-conversion functions 142
fields, comparing 157
file
CMDSEC 125, 136
data definitions 175
discard 255
ENCKEYS 135, 136
extract 14
ggserr.log 252
GLOBALS 25
header and version 295
parameter 25
trail, about 13
usrdecs.h 240
FILTER clause, TABLE or MAP 153, 226
filtering
data 153
DML operation types 158
transactions 100, 102
FILTERTABLE option, TRANLOGOPTIONS 97
functions
column-conversion 142
user exit 240

G

GENLOADSFILE parameter 211
GETAPPLOPS parameter 97
GETDELETES parameter 158
GETINSERTS parameter 158
GETREPLICATES parameter 97
GETUPDATEBEFORES parameter 158, 159
GETUPDATES parameter 158
GGFILEHEADER option, @GETENV 296
GGHEADER option, GETENV 159
ggmessage.dat file 173
GGS_BEFORE_AFTER_IND column 300
GGS_OP_TYPE column 300
GGS_TRANS_RBA column 300
GGSCI
security 136
using 23
global column mapping 149
global parameters 26
globalization support 41

GLOBALS file
creating 25
using with checkpoint table 186
groups
about 16
adding 187, 261
removing 198

H

HANDLECOLLISIONS parameter 201
header, file 295
header, record
description 298
overview 296
user token area 166
HELP command 7
high availability, planning for 77, 93
hot backup, for initial load 203

I

ID option, SQLEXEC 227
IF function 164
IGNORE DELETE option, FILTER clause 154
IGNORE INSERT option, FILTER clause 154
IGNORE option
EVENTACTIONS 243
REPERROR 169
SQLEXEC with ERROR 232
IGNORE UPDATE option, FILTER clause 154
IGNOREAPPLOPS parameter 97
IGNOREDELETES parameter 158
IGNOREINSERTS parameter 158
IGNOREREPLICATES parameter 97
IGNOREUPDATES parameter 158
INCLUDE parameter 238
INFO commands 247
initial data load
about 200
from file to database utility 209
from file to Replicat 204
using database utility 203
using direct bulk-load to SQL*Loader 219
using Oracle GoldenGate direct load 214
using Teradata load utilities 224
initializing transaction logs 268
INSERTALLRECORDS parameter 159, 171
INSERTDELETES parameter 159
inserts
creating from deletes or updates 159
into exceptions table 171
reversing to deletes 277
INSERTUPDATES parameter 159
IPv6 protocol 19

J

Japanese localization 177

K

key
database-generated values in 79, 95
encryption 134
name, supported characters 33
primary, in conflict resolution 94
KEYCOLS option, TABLE or MAP 201
KeyFieldComp record 304
KEYGEN 134

L

lag
estimating to determine number of parallel groups 184
monitoring 249
using heartbeat table to analyze 244
LAG command 247
LAGCRITICAL parameters 249
LAGINFO parameters 249
LAGREPORT parameters 249
language support 41

large objects, limitations on 142
latency
monitoring 249
viewing 247
library, macro 237
live reporting, configuring 43
live standby configuration, creating 77
loading data
from file to database utility 209
from file to Replicat 204
using database utility 203
using direct bulk load to SQL*Loader 219
using Oracle GoldenGate direct load 214
local trail, see *trail*
log
error 252
process 253
LOGEND option, SEND EXTRACT 269
login
Extract, specifying 191
Replicat, specifying 196
security 130, 136
looping, preventing 96

M

MACRO parameter 233
MACROCHAR parameter 234
macros
creating 233
excluding from report file 238
invoking from other macros 237
libraries 237
naming 234
running 235
tracing expansion 239
with parameters 235
Manager
about 15
autostart options 197
configuring and running 18
instances, number of 18
lag parameters 249
statistics, viewing 248
MAP parameter 142, 153
MAPEXCEPTION in MAP statement 170
mapping
columns 146
data types 152
rows 153
user tokens 166
with macros 236
MAXVARCHARLEN option, SQLEXEC 232
MEGABYTES option, ADD RMTTRAIL, ADD EXTTRAIL 191, 282, 283
messages, viewing 252
MGRPORT option, ADD EXTRACT 189
Microsoft SQL Server, see *SQL Server*
monitoring events and errors 247
MultiLoad, Teradata 224
MySQL, supported processing methods 10

N

name
supported characters in 33
names
with wildcards 37
network
communications, configuring 18
data encryption 133
instability 12
trusted zone configuration 138
NODBCHECKPOINT option, ADD REPLICAT 194, 283
NOEXTATTR option of DEFGEN 179
NOLIST parameter 238
NonStop, sending messages to 257
NOPARAMS option, SQLEXEC 227
NOPASSTHRU parameter 193
NULL function 157

null values, testing for 157, 164
numbers
comparing 157
mapping and transforming 152, 163
NUMBIN function 163
NUMSTR function 163

O

OBEY
command 24
parameter 26, 32
ODBC database, supported processing methods 10
ON DELETE option, FILTER clause 154
ON INSERT option, FILTER clause 154
ON UPDATE option, FILTER clause 154
online help, getting 7
online processing
about 16
changing 260
configuring 184
operations, SQL
history of 159
selecting and converting 158
statistics, viewing 247, 250
see also *transactions*
Oracle
passwords, encrypting 130
SQL*Loader initial load 209, 219
supported processing methods 10
Oracle GoldenGate
controlling 197
conversion functions 153, 165
messages file 173
overview and supported databases 9
record format 295, 296
user interfaces 23
ORACLE_SID, changing 274
overwrite recovery mode 14

P

parameter file
change synchronization
online extraction 191
online replication 195
creating and managing 25
DEFGEN 178
GLOBALS 25
initial load
bulk load 210, 211
direct bulk load 220, 222
direct load 215, 217
Replicat load 205, 206
Manager 20
Reverse utility 278
parameters
in
macros 235
SQL procedures and queries 227
retrieving from another file 32
substitution at runtime 32
using 25
verifying syntax 30
PARAMS option
ADD EXTRACT 30, 188
ADD REPLICAT 30, 194
MACRO 234
SQLEXEC 227
VAM 193
passive Extract 138
PASSIVE option, ADD EXTRACT 139, 188
pass-through data pump 12
PASSTHRU parameter 193
password
DEFGEN 178
encrypting 130
Extract 191
Replicat 196
patches, application 260
peer-to-peer configuration, creating 93
port number
dynamic list 19

port numbers
configuring 18
PORT option, ADD EXTRACT 189
procedures, see *stored procedures*
processes, Oracle GoldenGate
configurations 16
monitoring and statistics 168, 247
parallel 13, 261
PURGEOLDEXTRACTS parameter 20

Q

queries, executing through Oracle GoldenGate 226
QUERY clause, SQLEXEC 228

R

RAISEERROR option, FILTER 154
rate, processing 251
record, trail
about 296
formats 295, 296
recovery modes, about 14
redo threads
changing 273
specifying 188
remote trail, see *trail*
REPERROR parameter 168
Replicat
about 13
errors, handling 168
group, adding 194, 266
running
from GGSCI 196
transaction name 98
transaction, delaying 196
transactions
ignoring 100, 102
REPLICAT parameter 195
replicating data
about 13
bidirectionally 96
for change synchronization 184
for initial load 200
REPORT option
ADD EXTRACT 188
ADD REPLICAT 194
SEND commands 254
REPORT parameter 254
report, process
excluding macros 238
using 253
REPORTFILE option, ADD/SEND commands 254
reporting
errors during SQLEXEC processing 232
Extract processing 188
parameter file test 30
process events and errors 252, 253
Replicat processing 194
reporting configuration, creating 43
REPORTROLLOVER parameter 255
RESET option, REPERROR 169
Reverse utility 277
RMTHOST option, ADD EXTRACT 139, 189
RMTHOSTOPTIONS parameter 139
RMTTRAIL parameter 192
rows
inserting all 159
partitioning among processes in initial load 201
selecting and excluding 153

S

schemas, changing 269
scripts, batch and shell 24
secondary Extract process 12
security
data and passwords 125
GGSCI commands 136
sensitive data, excluding 158
selecting
columns 158
operations 158
rows 153
with stored procedures and queries 226
SEND commands 248
sensitive data, excluding 158

SERVLOG 258
SET EDITOR command 28
shell scripts, starting from 24
source database
attributes, changing 269
synchronizing
with another source database 93
with central target 12
with multiple targets 13
transaction history 159
source system
trails on 12
source tables
active during initial load 200
data definitions, creating 175
SOURCEDB parameter 232
source-definitions file, creating 175
SOURCEDEFS parameter 180
SOURCEISTABLE parameter 205, 210, 215, 220
special runs 16
SPECIALRUN option, ADD REPLICAT 216, 222
SPECIALRUN parameter
Replicat load 206
SQL Server
active-active support 93
bidirectional synchronization 98
bulk initial load 209
supported processing methods 10
SQL/MX, supported processing methods 10
SQLEXEC parameter 227
STARTUPVALIDATIONDELAY parameter 20
static Collector 16
statistics
operations processed 247
runtime 252
viewing for processes 168, 247
STATS command 247
STATUS command 247
stored procedures, using 226
STR* functions 163
strings
comparing and converting 163
substitution of parameter values 32
supplemental logging, changing attributes 271
Sybase, supported processing methods 10
synchronizing
initial load 200
syntax, verifying in parameter file 30
SYSLOG parameter 257
syslogs, Oracle GoldenGate messages in 256
system maintenance, performing 269

T

table
checkpoint
creating 185
specifying to Extract 97
exceptions 171
TABLE parameter 142, 147, 153
tables
adding to source database 269
DB2, reorganizing 275
dropping and recreating 273
mapping dissimilar 142
synchronizing changes 184
see also *source tables* and *target tables*
target systems
connections, initiating from 138
number of 63
target tables
inserting all records 159
populating 200
undoing changes 277
see also *tables*
TARGETDB parameter 232
target-definitions file, creating 175, 176
task, about 16
TCP/IP
data encryption 133
error handling 172
planning for unstable network 12
use of 15
TCPSOURCETIMER parameter 257
template, definitions 176

Teradata
load utility, using 224
supported processing methods 10
testing
column status 164
data 156, 165
for null values 157
for presence of a column 157
text editor, changing 28
THREADS option, ADD EXTRACT 188
threads, changing number of 273
throughput, data to target 250
timestamps
adjusting to match other systems 257
mapping 152
TimesTen, supported processing methods 10
tokens area of trail 302
TOKENS option, TABLE 166
tokens, user 166
trail
about 13
creating 190
encrypting 126
file size, changing 275
format 295
format and properties, returning 296
format, specifying 295
record format 295, 296
tokens, user 166
version of 295
version, specifying 295
TRANLOG option
ADD EXTRACT 187
TRANSABORT option, REPERROR 169
transaction log
as data source 187
intitializing 268
transactions
history of 159
identifying from source 17
ignoring 100, 102
preventing extraction of 96
Replicat, identifying and ignoring 96
skipping in trail 197
transforming data
in stages 12
with Oracle GoldenGate conversion functions 160
with user exits 240
troubleshooting, see *problem-solving*

U

UpdateComp record 303
UPDATECS option of DEFGEN 177
UPDATEDELETES parameter 159
updates
compressed 156
converting to inserts 159
creating from deletes 159
missing values, fetching 157, 231
reversing to previous state 277
simultaneous 105
USEDEFAULTS option, TABLE or MAP 148
USEIPV6 parameter 20
user
access to commands, controlling 136
excluding 100, 102
interfaces to Oracle GoldenGate 23
transaction, ignoring 100, 102
user exits, using 239
USERID parameter 20
usrdecs.h file 240
utilities
KEYGEN 134

V

VALONEOF function 165

values
comparing before and after 157
converting in columns 163
invalid, null, missing 164
making available for filters 156
null, testing for 157
VAM option, ADD EXTRACT 188
VAM parameter 193
VAMTRAILSOURCE option, ADD EXTRACT 188
verifying parameter files 30
version, displaying 295
version, of trail or extract file 295
VIEW GGSEVT command 253
VIEW PARAMS command 30
viewing
command permissions 136
encryption file 135, 136
errors and statistics 168, 247
macro expansion 239
parameters 30
volume statistics, getting 250

W

warnings
as event action during processing 244
for missing columns during filtering 157
viewing 252
WARNRATE parameter 255
WHERE clause
for record selection 156
WILDCARDRESOLVE parameter 270
wildcards
in command security file 136
in commands 23
using 37
when adding tables 270
Windows CP932 177