Oracle® GoldenGate

Windows and UNIX Administrator's Guide

11*g* Release 1 Patch Set 1 (11.1.1.1)

E21513-01

April 2011

Oracle GoldenGate Windows and UNIX Administrator’s Guide 11g Release 1 Patch Set 1 (11.1.1.1)

E21513-01

Contents

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

To get help for either Oracle GoldenGate Director Client or Oracle GoldenGate Director Web, 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

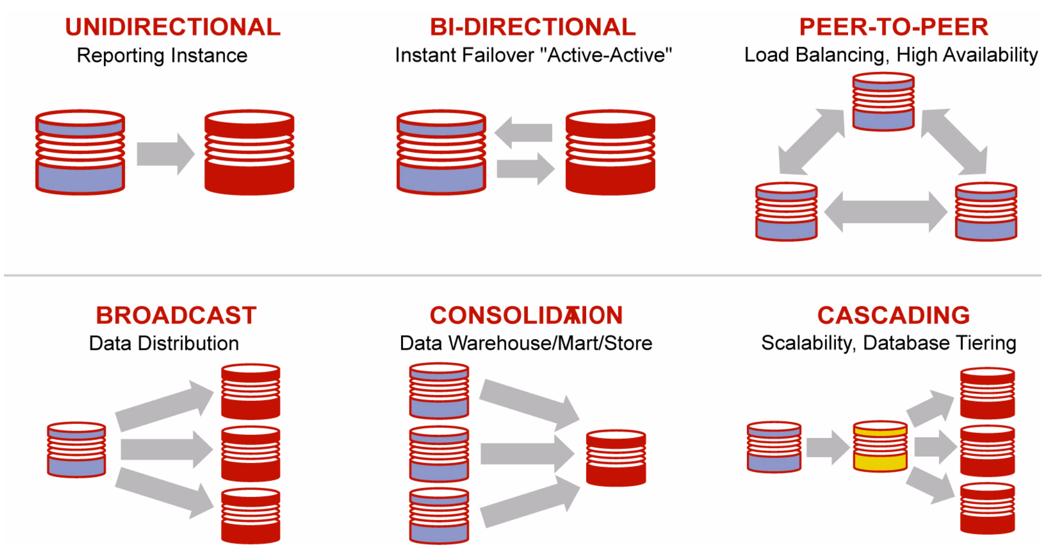
Oracle GoldenGate supported processing methods and databases

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


UNIDIRECTIONAL
Reporting Instance
BI-DIRECTIONAL
Instant Failover "Active-Active"
PEER-TO-PEER
Load Balancing, High Availability
BROADCAST
Data Distribution
CONSOLIDATION
Data Warehouse/Mart/Store
CASCADING
Scalability, Database Tiering

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)
c-tree***	X		X
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.

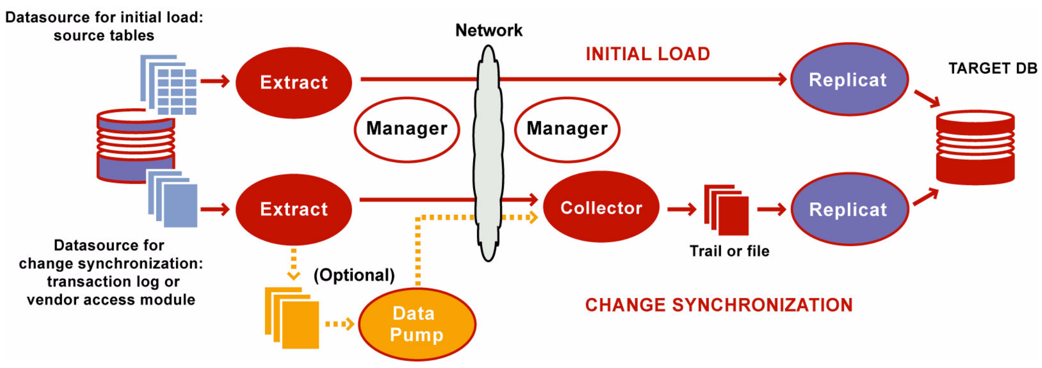
Overview of the Oracle GoldenGate architecture

Oracle GoldenGate is composed of the following components:

- Extract
- 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 replication of ongoing database changes. 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 a current set of data directly from their source objects.
- **Change synchronization:** To keep source data synchronized with another set of data, Extract captures changes made to data (typically transactional inserts, updates, and deletes) after the initial synchronization has taken place. DDL changes and sequences are also extracted, if supported for the type of database that is being used.

When processing data changes, Extract obtains the data from a *data source* that can be one of the following.

- The *database recovery logs or transaction logs* (such as the Oracle redo logs or SQL/MX audit trails). The actual method of obtaining the data from the logs varies depending on the databast type.
- A *third-party capture module*. This method provides a communication layer that passes data changes 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 changes and pass it to Extract.

Extract captures all of the changes that are made to objects that you configure for synchronization. Extract stores the changes until it receives commit records or rollbacks for the transactions that contain them. When a rollback is received, Extract discards the data for that transaction. When a commit is received, Extract sends the data for that transaction to the trail for propagation to the target system. All of the log records for a 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, one process could continuously extract transactional data changes and stream them to a decision-support database, while another process performs batch extracts for periodic reporting. Or, two Extract processes could extract and transmit in parallel to two Replicat processes (with two trails) to minimize target latency when the databases are large. To differentiate among different processes, you assign each one a group name (see "Overview of groups" on page 18).

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 data to a remote trail on the target. In a typical configuration that includes 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 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.

Like a primary Extract group, a data pump can be configured for either online or batch processing. It 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 data that Extract continuously extracts into memory. If the network or the target system becomes unavailable, that 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 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.

If your requirements preclude the use of a data pump, you can still configure Oracle GoldenGate without one. Oracle GoldenGate supports many different configurations. See the configuration chapters in this guide to find the one that is best suited to your environment.

Overview of Replicat

The Replicat process runs on the target system. Replicat reads extracted data changes and DDL changes (if supported) that are specified in the Replicat configuration, and then it replicates them to the target database. You can configure Replicat in one of the following ways:

- **Initial loads:** For initial data loads, Replicat can apply data to target objects or route them to a high-speed bulk-load utility.
- **Change synchronization:** To maintain synchronization, Replicat applies extracted data changes to target objects using a native database interface or ODBC, depending on the database type. Replicated DDL and sequences are also applied, if supported for the type of database that is being used. To preserve data integrity, Replicat applies the replicated changes 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. Each set of processes handles different objects. To differentiate among processes, you assign each one a group name (see “Overview of groups” on page 18).

You can delay Replicat so that it waits a specific amount of time before applying data 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 the captured changes temporarily on disk in a series of files called a *trail*. A trail can exist on the source or target system, or on an intermediary system, 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 16). 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 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 process writes to a trail. Only one Extract process can write to a trail.

Processes that read the trail are:

- Data-pump Extract: Extracts data from a local trail for further processing, if needed, and transfers it to the target system or to the next Oracle GoldenGate process downstream in the Oracle GoldenGate configuration.
- Replicat: Reads a trail to apply change data to the target database.

Trail maintenance

Trail files are created as needed during processing, and they are aged automatically to allow processing to continue without interruption for file maintenance. By default, trails are stored in the dirdat sub-directory of the Oracle GoldenGate directory.

By default, each file in a trail is 10 MB in size. All file names in a trail begin with the same two characters, which you assign when you create the trail. As files are created, each name is appended with a unique, six-digit serial (sequence) number from 000000 through 999999, for example c:\ggs\dirdat\tr000001. When the trail 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.

How processes write to a trail

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.

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 format

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.

Each data record in a trail file also contains a header area, as well as a data area. The record header contains information about the transaction environment, and the data area contains the actual data values that were extracted. For more information about the trail record format, see Appendix 1.

File versioning

Because all of the Oracle GoldenGate processes are decoupled and thus can be of different Oracle GoldenGate versions, each trail file or extract file has a version that is stored in the file header. By default, the version of a trail is the current version of the process that created the file. 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).

Overview of extract files

When processing a one-time run, such as an initial load or a batch run that synchronizes transactional changes (see page 18), Oracle GoldenGate stores the extracted changes in an *extract file* instead of a trail. The extract file typically is a single file but 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. These checkpoints ensure that data changes that are marked for synchronization actually are extracted by Extract and replicated 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. Replicat creates checkpoints for its position in the trail.

A checkpoint system is used by Extract and Replicat processes that operate continuously, but it is not required by Extract and Replicat processes that run in batch mode (see page 18). A batch process can be re-run from its start point, whereas continuous processing requires the support for planned or unplanned interruptions that is provided by checkpoints.

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. This ensures 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. You can optionally configure Replicat to use this file as its sole checkpoint store, and not use a checkpoint table at all. In this mode, however, there can be cases where the checkpoint in the file is not consistent with what was applied after a database recovery, if the failure either rolled back or rolled forward a transaction that was considered applied by Replicat. The checkpoint table guarantees consistency after recovery.

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:

- Monitor and restart Oracle GoldenGate processes.
- Issue threshold reports, for example when throughput slows down or when synchronization latency increases.
- Maintain trail files and logs.
- Allocate data storage space.
- Report errors and events.
- Receive and route requests from the user interface.

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, see Chapter 2.

Overview of Collector

Collector is a process that runs in the background on the target system. Collector receives extracted database changes that are sent across the TCP/IP network, and it writes them to a trail or extract file. Typically, Manager starts Collector automatically when a network connection is required. When Manager starts Collector, the process is known as a *dynamic* Collector, and Oracle GoldenGate users generally do not interact with it. However, you can run Collector manually. This is known as a *static* Collector. Not all Oracle GoldenGate configurations use a Collector process.

When a dynamic Collector is used, it can receive information from only one Extract process, so there must be a dynamic Collector for each Extract that you use. When a static Collector is used, several Extract processes can share one Collector. However, a one-to-one ratio is optimal. The Collector process terminates when the associated Extract process terminates.

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.

Overview of processing methods

Oracle GoldenGate can be configured for the following purposes:

- A static extraction of selected data records from one database and the loading of those records to another database.
- Online or batch extraction and replication of selected transactional data changes 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.

For these purposes, Oracle GoldenGate supports the following processing modes.

- An *online* process runs until stopped by a user. Online processes maintain recovery checkpoints in the trail so that processing can resume after interruptions. You can use online processes to continuously extract and replicate transactional changes and DDL changes (where supported).
- A *batch run*, or *special run*, process extracts or replicates database changes that were generated within known begin and end points. For special runs, Oracle GoldenGate does not maintain checkpoints. Should a process fail, the job can be started over, using the same begin and end points. You can use a special run to process a batch of database changes (such as to synchronize source and target objects once a day rather than continuously) or for an initial data load.
- A *task* is a special type of batch run process and is used for certain initial load methods. A task is a configuration 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.

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. A group name can be as follows:

Table 2 Permissible group names

- 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.
- Do not place a numeric value at the end of a group name, such as fin1, fin10, and so forth. You can place a numeric value at the beginning of a group name, such as 1_fin, 1fin, and so forth.

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. 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 particular 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.

Extract writes a normalized form of the CSN to external storage such as the trail files and the checkpoint file. There, the CSN is represented as a hex string of bytes. In normalized form, the first two bytes represent the database platform, and the remainder of the string represents the actual unique identifier.

The CSN is also included in report output, error messages, and command input and output (as appropriate) in human-readable, display form that uses native character encoding. In this form, the database type is not included, but it can be supplied separately from the identifier.

Table 3 Oracle GoldenGate CSN values per database[1]

Database	CSN Value

c-tree	`<log number>.<byte offset>`
	Where:
	◆ <log number> is the 10-digit decimal number of the c-tree log file padded with leading zeroes.
	◆ <byte offset> is the 10-digit decimal relative byte position from the beginning of the file (0 based) padded with leading zeroes.
	Example:
	`0000000068.0000004682`

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`

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`

Table 3 Oracle GoldenGate CSN values per database[1] (continued)

Database	CSN Value

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`

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`

Table 3 Oracle GoldenGate CSN values per database[1] (continued)

Database	CSN Value

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:
	`0x0800000000000000D700000021`

TimesTen	There is no CSN for TimesTen, because extraction (capture) is not supported by Oracle GoldenGate for this database.

[1] 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.

CHAPTER 2

Configuring the Manager process

Overview of the Manager process

To configure and run Oracle GoldenGate, a Manager process must be running on the source and target systems. The Manager process performs the following functions:

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

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

Configuring Manager

To configure Manager, create a parameter file by following these steps. If you installed Oracle GoldenGate on a UNIX cluster, configure the Oracle GoldenGate Manager process within the cluster application as directed by the vendor's documentation, so that Oracle GoldenGate will fail over properly with the other applications.

To configure Manager

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

   ```
   EDIT PARAMS MGR
   ```

3. Add the following parameter to specify the Manager port number.

   ```
   PORT <port_number>
   ```

PORT defines the port number on which Manager runs on the local system. Observe these guidelines:

- The default port number is 7809.
- You must specify either the default port number or a different one.
- Each Manager instance on a system must use a different port number.
- The port must be unreserved and unrestricted. GGSCI uses this port to request Manager to start processes. The Extract process uses this port to request Manager to start a remote Collector process or an initial-load Replicat process.

- PORT is the only required Manager parameter.

  NOTE The port number also must be specified with the MGRPORT argument of the Extract parameter RMTHOST.

4. Enter any of the optional Manager parameters documented in the Oracle GoldenGate *Windows and UNIX Reference Guide*, then save and close the file.

Recommended parameters

The following parameters are optional, but recommended for the Manager process. For more information and syntax, see the *Windows and UNIX Reference Guide*.

- USERID: Required if using Oracle GoldenGate DDL support, specify the Manager user and password with the .
- DYNAMICPORTLIST: Use to specify up to 256 unreserved, unrestricted ports for dynamic TCP/IP communications between the source and target systems. The Collector, Replicat, and GGSCI processes will use these ports. In the absence of DYNAMICPORTLIST, Manager tries to start Collector on port 7840. If 7840 is not available, Manager increments by one until it finds an available port.
- DYNAMICPORTREASSIGNDELAY: Controls how long Manager waits to assign a port that was assigned before.
- AUTOSTART: Starts Extract and Replicat processes when Manager starts. This can be useful, for example, if you want Oracle GoldenGate activities to begin immediately when you start the system, assuming Manager is 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.
- PURGEOLDEXTRACTS: Purges trail files when Oracle GoldenGate has finished processing them. Without using PURGEOLDEXTRACTS, no purging is performed, and trail files can consume significant disk space.Using PURGEOLDEXTRACTS as a Manager parameter is preferred over using the Extract or Replicat version of PURGEOLDEXTRACTS.

  NOTE When using PURGEOLDEXTRACTS, do not permit trail files to be deleted by any user or program other than Oracle GoldenGate. It will cause PURGEOLDEXTRACTS to function improperly.

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
- GGSCI
- 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 line

To run Manager from the command shell of the operating system, use the following syntax.

```
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
```

On Windows systems, you can use the BOOTDELAYMINUTES parameter to specify how long after system boot time Manager delays until it starts its processing activities.

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, Linux, and z/OS using USS, Manager must be stopped by using the STOP MANAGER command in GGSCI.

```
STOP MANAGER [!]
```

  Where: ! stops Manager without user confirmation.

- On Windows, if Manager is installed as a service, you can stop it from the Services applet. See the Windows documentation or your system administrator.
- In a Windows cluster, Manager must only be stopped by taking the Manager resource offline by using the Cluster Administrator. If you attempt to stop the Manager service from the GGSCI interface, the cluster monitor interprets it as a resource failure and will attempt to bring the resource online again. If a stop request is repeatedly submitted from the GGSCI interface, and the restart threshold of the Manager cluster resource is exceeded, the cluster monitor marks the Manager resource as failed.
- 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.

CHAPTER 3

Getting started with Oracle GoldenGate

Running the user interfaces

Use any of the following methods to control and monitor processing:

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

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*.

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.

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. Use the following syntax:

```
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 an ASCII file that is read by an associated process. Oracle GoldenGate uses two types of parameter files: a GLOBALS file and runtime parameter files.

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. Once set, GLOBALS parameters are rarely changed, and there are far fewer of them than runtime parameters.

A GLOBALS parameter file is required only in certain circumstances and, when used, must be created from the command shell 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 an ASCII 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 33.

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.

Figure 4 Sample Extract parameter file

```
EXTRACT capt
USERID ggs, PASSWORD *********
DISCARDFILE /ggs/capt.dsc, PURGE
RMTHOST sysb, MGRPORT 7809
RMTTRAIL /ggs/dirdat/aa
TABLE fin.*;
TABLE sales.*;
```

Figure 5 Sample Replicat parameter file

```
REPLICAT deliv
USERID ggs, PASSWORD ****
SOURCEDEFS /ggs/dirdef/defs
DISCARDFILE /ggs/deliv.dsc, PURGE

GETINSERTS
MAP fin.account, TARGET fin.acctab,
COLMAP (account = acct,
balance = bal,
branch = branch);

MAP fin.teller, TARGET fin.telltab,
WHERE (branch = "NY");

IGNOREINSERTS
MAP fin.teller, TARGET fin.telltab,
WHERE (branch = "LA");
```

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.

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
```

NOTE On Linux, the group and parameter file names must be all uppercase or all lowercase. Usage of mixed case file names result in errors when starting the process.

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 ggs, PASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH, &
  ENCRYPTKEY superx128
  RMTHOST sysb, MGRPORT 8040
  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.

5. Save and close the file.

To create a parameter file with a text editor

1. Create a new file in the text editor.
2. 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 (--).
3. 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 ggs, PASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH, &
     ENCRYPTKEY superx128
     RMTHOST sysb, MGRPORT 8040
     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.

4. Save and close the file. When creating a parameter file outside GGSCI, 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.
   - Save the parameter file in the dirprm directory of the Oracle GoldenGate installation directory.

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 19.

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.

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.

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.

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 269.

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 c:\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

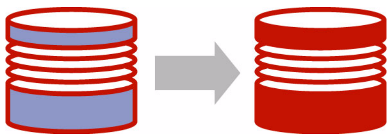
For more information about Oracle GoldenGate parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide.*

CHAPTER 4

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 37
- "Creating a reporting configuration with a data pump on the source system" on page 39
- "Creating a reporting configuration with a data pump on an intermediary system" on page 42
- "Creating a cascading reporting configuration" on page 46

Multiple targets

You can send data to multiple reporting targets. See "Using Oracle GoldenGate for real-time data distribution" on page 53.

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 238
- "Customizing Oracle GoldenGate processing" on page 262

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 79.

Additional information

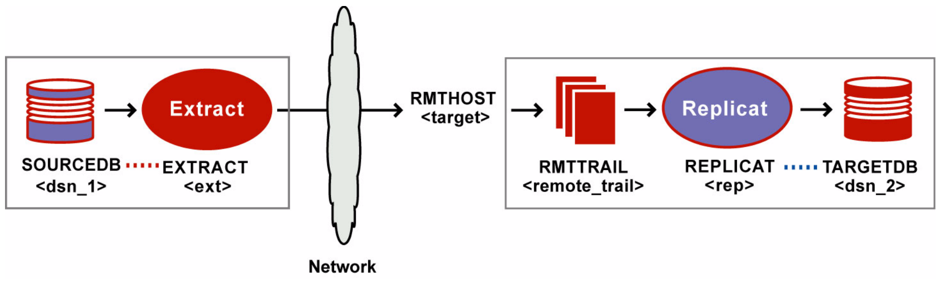
- For additional system and database configuration requirements, see the Oracle GoldenGate *Installation and Setup Guide* for your database type.
- For information about additional requirements for Teradata Extract configurations, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see "Configuring online change synchronization" on page 120.
- 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, BEGIN <time> [, THREADS <n>]
```

   - Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
   -- Specify the name or IP address of the target system:
   RMTHOST <target>, MGRPORT <portnumber>
   -- Specify the remote trail on the target system:
   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 121. 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 id>[, PASSWORD <pw>]]
-- 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 the source system

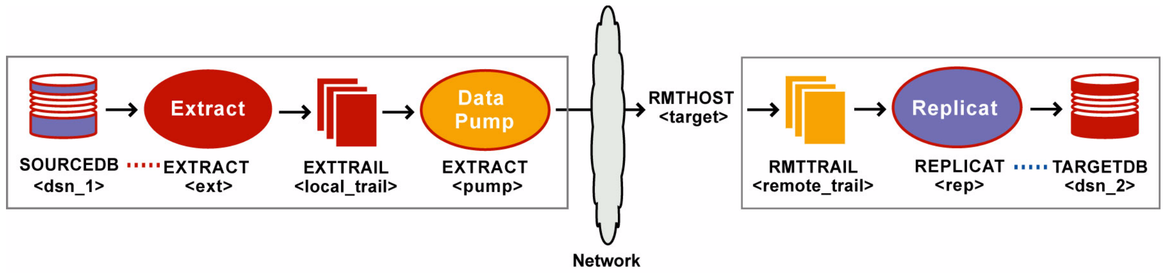
You can add a data pump on the source system to isolate the primary Extract from TCP/IP functions, to add storage flexibility, and to offload the overhead of filtering and conversion processing from the primary Extract.

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, BEGIN <time> [, THREADS <n>]
```

❍ Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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 as needed for the database:
[SOURCEDB <dsn_1>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of the target system:
RMTHOST <target>, MGRPORT <portnumber>
-- Specify the remote trail on the target system:
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 121.
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 id>[, PASSWORD <pw>]]
-- 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. There is no need to have a database on the intermediary system.

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.

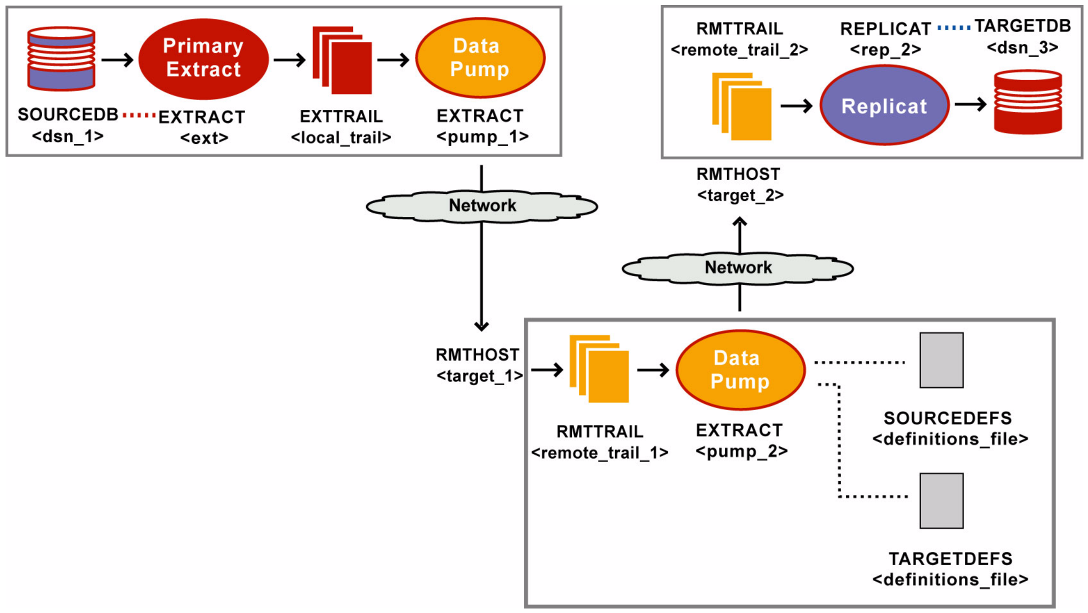
NOTE The data pump on the source system is optional, but will help to protect against data loss in the event of a network outage.

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 11.

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, BEGIN <time> [, THREADS <n>]
```

   ❍ Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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 as needed for the database:
[SOURCEDB <dsn_1>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of the intermediary system:
RMTHOST <target_1>, MGRPORT <portnumber>
-- Specify the remote trail on the intermediary system:
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>
-- 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 the name or IP address of the target system:
RMTHOST <target_2>, MGRPORT <portnumber>
-- Specify the remote trail on the target system:
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 121.

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 id>[, PASSWORD <pw>]]
-- 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.

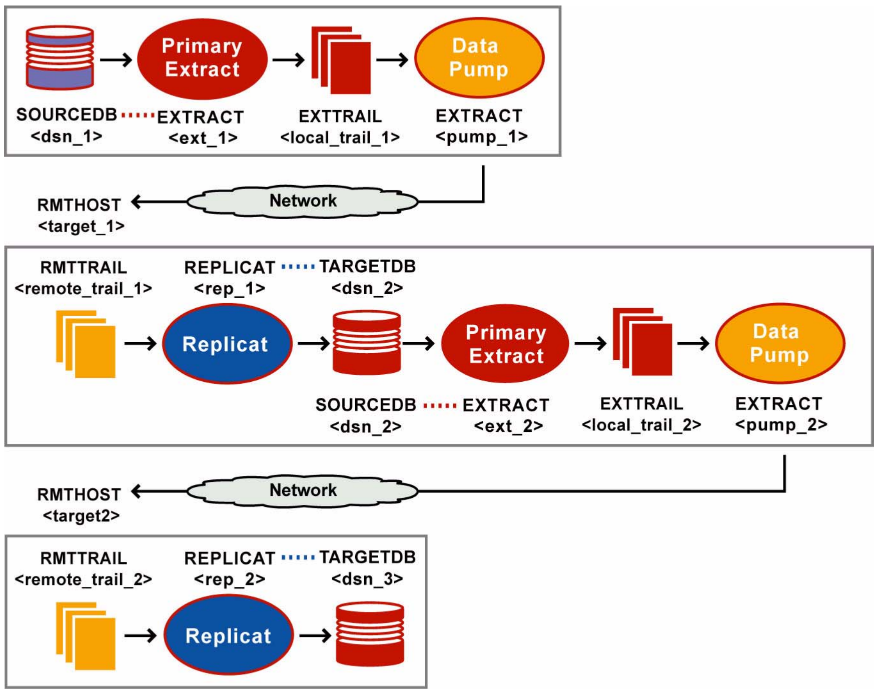
In this configuration:

- 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 42.

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, BEGIN <time> [, THREADS <n>]
   ```

   ❍ Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
   -- Specify the local trail that this Extract writes to:
   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 as needed for the database:
[SOURCEDB <dsn_1>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of second system in cascade:
RMTHOST <target_1>, MGRPORT <portnumber>
-- Specify the remote trail on the second system:
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 121.
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 id>[, PASSWORD <pw>]]
-- 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, BEGIN <time> [, THREADS <n>]
```

   - Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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 as needed for the database:
[SOURCEDB <dsn_2>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of third system in cascade:
RMTHOST <target_2>, MGRPORT <portnumber>
-- Specify the remote trail on the third system:
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 121.
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.

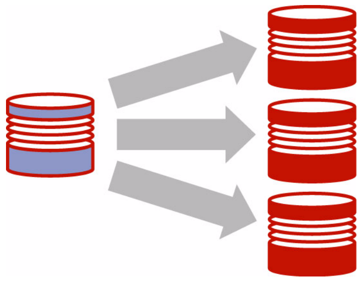
    ```
    -- Identify the Replicat group:
    REPLICAT <rep_2>
    -- State whether or not source and target definitions are identical:
    SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
    -- Specify database login information as needed for the database:
    [TARGETDB <dsn_3>,] [USERID <user id>[, PASSWORD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

CHAPTER 5

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 79.

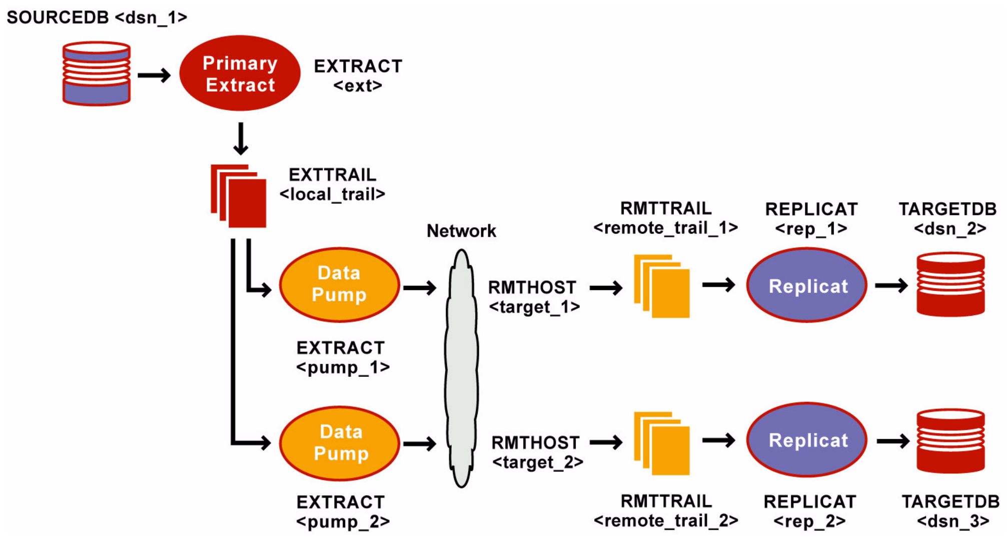
Additional information

- 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 120.
- 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, BEGIN <time>, [, THREADS]
```

   ❍ Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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 as needed for the database:
[SOURCEDB <dsn_1>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of the first target system:
RMTHOST <target_1>, MGRPORT <portnumber>
-- Specify the remote trail on the first target system:
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_1>,][USERID <user>[, PASSWORD <pw>]]
-- Specify the name or IP address of the second target system:
RMTHOST <target_2>, MGRPORT <portnumber>
-- Specify the remote trail on the second target system:
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 121.
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 id>[, PASSWORD <pw>]]
-- 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 id>[, PASSWORD <pw>]]
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery:
MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
```

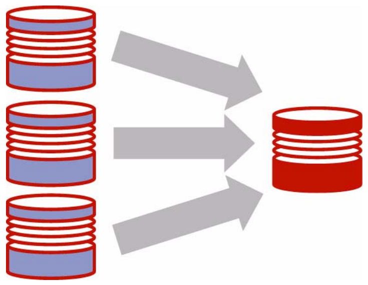
❍ You can use any number of MAP statements for any given Replicat group. All MAP statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

CHAPTER 6

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 79.

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*.

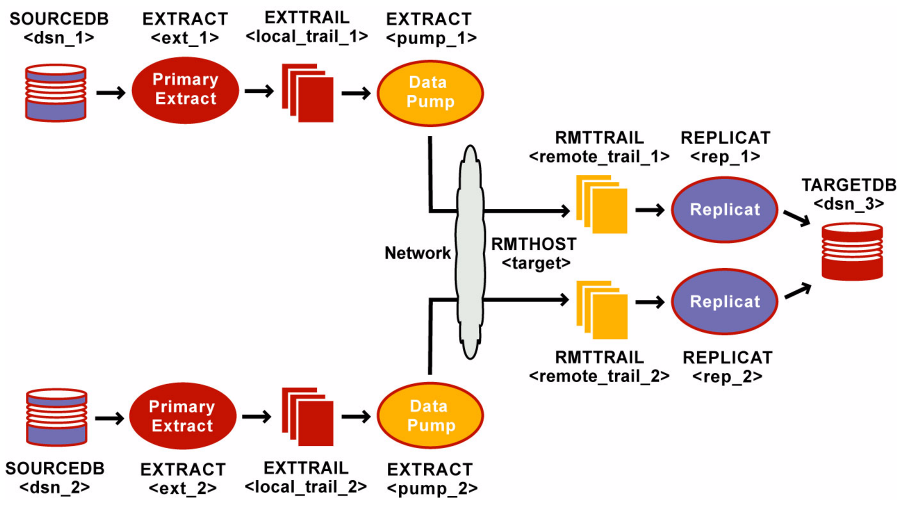
Additional information

- 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 120.
- 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, BEGIN <time> [, THREADS <n>]
   ```

   Extract_2

   ```
   ADD EXTRACT <ext_2>, TRANLOG, BEGIN <time> [, THREADS <n>]
   ```

   ❍ Use TRANLOG as the data source option. For DB2 on Z/OS, specify the bootstrap data set (BSDS) name following TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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>]]
-- Specify the local trail that this Extract writes to:
EXTTRAIL <local_trail_2>
-- 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>]]
-- Specify the name or IP address of the target system:
RMTHOST <target>, MGRPORT <portnumber>
-- Specify the remote trail on the target system:
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>]]
-- Specify the name or IP address of the target system:
RMTHOST <target>, MGRPORT <portnumber>
-- Specify the remote trail on the target system:
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 id>[, PASSWORD <pw>]]
    -- Specify error handling rules:
    REPERROR (<error>, <response>)
    -- Specify tables for delivery:
    MAP <owner>.<table>, TARGET <owner>.<table>[, DEF <template name>];
    ```

    Replicat_2

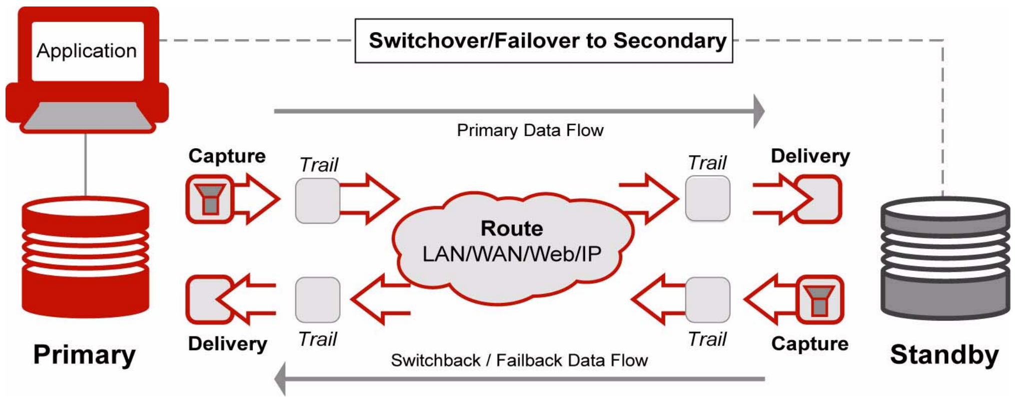
    ```
    -- Identify the Replicat group:
    REPLICAT <rep_2>
    -- State whether or not source and target definitions are identical:
    SOURCEDEFS <full_pathname> | ASSUMETARGETDEFS
    -- Specify database login information as needed for the database:
    [TARGETDB <dsn_3>,] [USERID <user id>[, PASSWORD <pw>]]
    -- 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
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 8 on page 79.

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*.

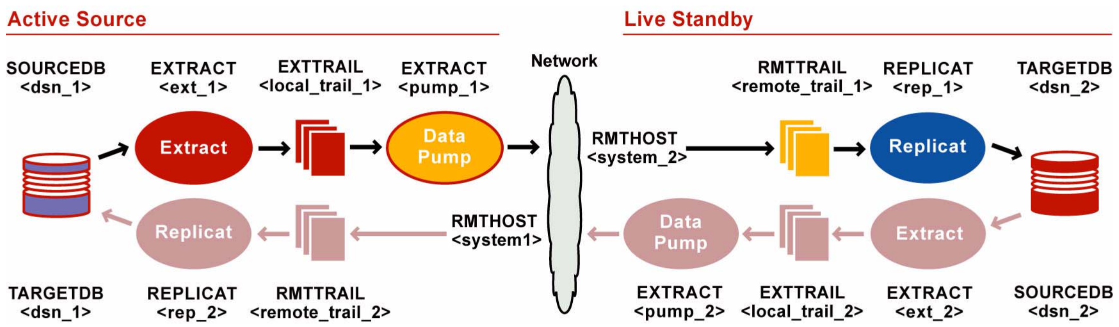
Additional information

- 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 120.
- 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 121.
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*.

   ```
   ADD EXTRACT <ext_1>, TRANLOG, BEGIN <time> [, THREADS <n>]
   ```

   - Use TRANLOG as the data source.
   - For DB2 on Z/OS, specify the bootstrap data set (BSDS) name after TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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>]]
   -- Specify the name or IP address of the standby system:
   RMTHOST <system_2>, MGRPORT <portnumber>
   -- Specify the remote trail on the standby system:
   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 121.
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 id>[, PASSWORD <pw>]]
   -- 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, BEGIN <time> [, THREADS <n>]
   ```

   ❍ Use TRANLOG as the data source.
   ❍ For DB2 on Z/OS, specify the bootstrap data set (BSDS) name after TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
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>]]
-- Specify the name or IP address of the active source system:
RMTHOST <system_1>, MGRPORT <portnumber>
-- Specify the remote trail on the active source system:
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 id>[, PASSWORD <pw>]]
   -- 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, BEGIN <time> [, THREADS <n>]
   ```

   - Use TRANLOG as the data source for all databases.
   - For DB2 on Z/OS, specify the bootstrap data set (BSDS) name after TRANLOG.

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>
   ```

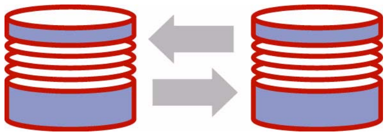
5. On the **primary** system, allow users to access the applications.

CHAPTER 8

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:

- c-tree
- 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. See Chapter 14 on page 141 for configuration information.

Number of databases

The most common peer-to-peer solution to improve scalability and disaster tolerance uses two identical databases. Any more than that, and resynchronization without downtime becomes extremely complex, and it makes the conflict-resolution routines more complex to design and manage.

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. For more information about keys, see the Oracle GoldenGate installation guide for your database.

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.

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 both 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. See "Managing conflicts" on page 91.

Additional information

- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see "Configuring online change synchronization" on page 120.
- 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)

By default, Extract ignores SQL operations that are generated by Replicat if the database is anything other than Teradata. The parameters that control this functionality are:

- 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

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.

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.

c-tree

Extract automatically identifies Replicat transactions that are applied to a c-tree database.

Oracle

(**Oracle 10g and later**) 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>
  ```

(**Oracle 9i and earlier**) Create a trace table with the ADD TRACETABLE command in GGSCI.

To create an Oracle trace table

Perform the following steps on each source database.

1. Run GGSCI.
2. In GGSCI, issue the following command to log into the database.

   ```
   DBLOGIN USERID <user>, PASSWORD <password>
   ```

   - <user> is the database user that is assigned to the Extract process as listed in the USERID parameter, and <password> is that user's password. The table must be created in the schema that belongs to this user.

3. In GGSCI, issue the following command to create the trace table.

   ```
   ADD TRACETABLE [<owner>.<table name>]
   ```

   - <owner>.<table name> is required only if using a name other than the default of GGS_TRACE. The owner must be the database user that is assigned to the Extract process as listed in the USERID parameter. Whenever possible, use the default name.

To associate the Oracle trace table with Extract and Replicat processes

If you created an Oracle trace table with a name other than GGS_TRACE, include the following parameter statement in the Extract and Replicat parameter files, preceding any TABLE or MAP parameters.

```
TRACETABLE [<owner>.<table name>]
```

Where: `<owner>.<table name>` is required only if using a name other than the default of GGS_TRACE. The owner must be the Extract user that is specified with the USERID parameter. Whenever possible, use the default name.

What the Oracle trace table does

If used, TRACETABLE must appear in both the Extract and Replicat parameter files.

- In the Replicat parameter file, TRACETABLE causes Replicat to write an operation to the trace table at the beginning of each transaction.
- In the Extract parameter file, TRACETABLE causes Extract to identify as a Replicat transaction any transaction that begins with an operation on the trace table.

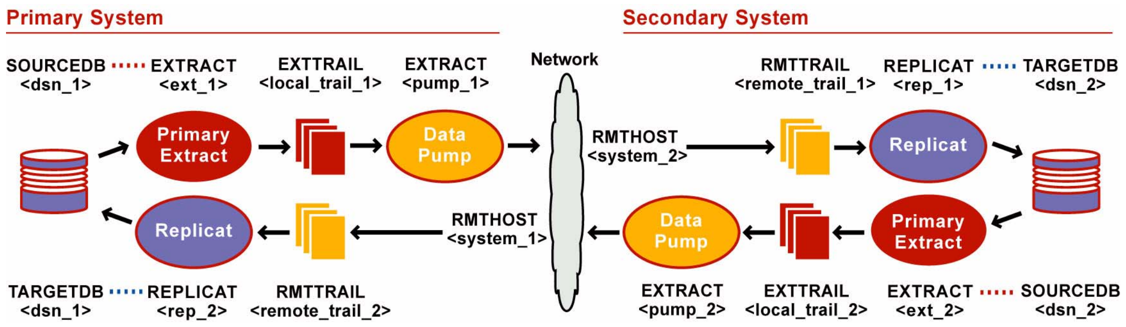
NOTE Although not recommended, if a trace table and EXCLUDEUSER or EXCLUDEUSERID are both used for the same Extract configuration, they will both function correctly. Whatever transactions are specified for each one will be processed according to the rules of GETREPLICATES or IGNOREREPLICATES.

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 304 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 121.
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, BEGIN <time> [, THREADS <n>]
```

   - Use TRANLOG as the data source.
   - For DB2 on Z/OS, specify the bootstrap data set (BSDS) name after TRANLOG.

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>]]
-- Specify the local trail that this Extract writes to:
EXTTRAIL <local_trail_1>
--
-- Exclude Replicat transactions. Uncomment ONE of the following:
-- DB2 z/OS and LUW, and Sybase:
```

```
-- TRANLOGOPTIONS EXCLUDEUSER <Replicat_user>
-- 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>
--

-- Specify API commands if Teradata:
VAM <library name>, PARAMS (“<param>” [, “<param>”] [, ...])
-- Capture before images for conflict resolution:
GETUPDATEBEFORES
-- 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>]]
   -- Specify the name or IP address of the secondary system:
   RMTHOST <system_2>, MGRPORT <portnumber>
   -- Specify the remote trail on the secondary system:
   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 id>[, PASSWORD <pw>]]
   -- Specify error handling rules:
   REPERROR (<error>, <response>)
   -- Specify tables for delivery and call conflict-resolution routines:
   MAP <owner>.*, TARGET <owner>.*, SQLEXEC (<SQL specification>);
   -- 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, BEGIN <time> [, THREADS <n>]
   ```

   ❍ Use TRANLOG as the data source.
   ❍ For DB2 on Z/OS, specify the bootstrap data set (BSDS) name after TRANLOG.

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>]]
-- 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, and Sybase:
-- TRANLOGOPTIONS EXCLUDEUSER <Replicat_user>
-- 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:
GETUPDATEBEFORES
-- 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>]]
-- Specify the name or IP address of the primary system:
RMTHOST <system_1>, MGRPORT <portnumber>
-- Specify the remote trail on the primary system:
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 id>[, PASSWORD <pw>]]
-- Specify error handling rules:
REPERROR (<error>, <response>)
-- Specify tables for delivery and call conflict-resolution routines:
MAP <owner>.*, TARGET <owner>.*, SQLEXEC (<SQL specification>);
-- 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 replicated insert attempts to add a row that already exists in the target.
- the before image of a replicated update does not match the current row in the target.
- a replicated delete attempts to remove a row that does not exist in the target.

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.

Minimizing the potential for 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.

Resolving conflicts through Oracle GoldenGate

Because conflict-resolution routines must be customized to specific applications and business rules, Oracle GoldenGate does not provide default procedures for handling conflicts. Oracle GoldenGate can interact with custom conflict-resolution routines that you write to satisfy your business rules.

To configure Oracle GoldenGate to execute and manage conflict-resolution routines

See "Conflict detection and resolution examples" on page 96 for an example of how to combine the following configuration elements into an automated conflict-resolution system.

1. Create the conflict-resolution routines. For Oracle GoldenGate, it works best to supply the rules as a stored SQL procedure or query. See "Guidelines for writing successful routines" on page 93 and "Methods for resolving conflict" on page 94.
2. Call the conflict resolution routine from the Oracle GoldenGate process by using the SQLEXEC parameter in your MAP statements. In the same MAP statement, you can use a FILTER clause to filter for a specific condition based on the output of the SQL, as needed. You can use multiple MAP statements in this manner to create different rules for

different tables, and these rules can be as simple or as complex as needed. For more information about MAP options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

3. Use the REPERROR parameter to assign error-handling rules to specific error numbers. You can assign default rules and/or error-specific rules.
4. (Optional) If you specify a response of EXCEPTION for REPERROR, you can use the MAP parameter with EXCEPTIONSONLY or MAPEXCEPTION to log operations that caused conflicts to an exceptions table. Within the same transaction that triggered the conflict, you can query the data from the exceptions table to calculate adjustments to the outcome of the transaction, to process the conflict manually, or to use a custom program to resolve the conflicts. For example, you can use an exceptions table to keep track of inventory items that become unavailable due to conflicting purchase transactions, so that at the end of the transaction that lost the conflict, the invoice can be adjusted for the item that is out of stock. As a historical tool, an exceptions table makes it easy to find out what conflicts occurred and how they were resolved. It also can help you with troubleshooting your conflict resolution routines.

The flexibility of the Oracle GoldenGate configuration objects allow you to create different rules for different tables or groups of tables. For example, you can discard a replicated operation and keep the existing values in some target tables, but overwrite the existing values with the replicated ones for other target tables.

For additional information about using exceptions mapping, see “Handling Replicat errors during DML operations” on page 110.

To create an exceptions table for conflict resolution

Create an exceptions table the same way that you would create any other table in the database. The following are suggested columns to include in an exceptions table.

- The primary key value on the exceptions table.
- The operation type.
- The name of the table for which the conflict occurred.
- The primary key value for the row that was being modified. This lets you look up the row again to verify data when needed.
- An image of the data that was applied to the target row.
- The before-image of the source record that was replicated. This is useful for comparing the old image against an overwritten image to determine if the conflict-resolution routine was correct in its resolution. This is important when resolving quantitative values such as those in inventories.
- The before-image of the data that was in the record on the target side, which was overwritten. This is helpful for notification and for resolving discrepancies that may not have been solved by the resolution routines.
- The time that the conflict was resolved.
- The timestamp of the record that won the conflict, if using timestamp-based resolution.
- The timestamp that was in the original target row before it was overwritten, if using timestamp-based resolution. Ensuring that this value is newer or older than the timestamp of the record that won (depending on your business rule) provides proof that the timestamp-based routine was successful.

- The name of the system, database, or user that won the conflict, if using trusted-source resolution.
- An indicator (Y/N) stating whether or not a row was discarded to the discard table, if used.
- An indicator (Y/N) stating whether or not the user should be notified of the conflict results.
- An indicator (Y/N) stating whether or not the user was notified of the conflict results.

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.

Guidelines for writing successful routines

Use the same 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.

Use before images to make comparisons

Oracle GoldenGate provides a mechanism for obtaining the values of a row as it was before a modification was made. Use the before image of each row in your conflict-resolution routines to make comparisons for update operations.

The before image allows you to compare the existing image of one or more columns in the target table with the before image of those columns in a modification that is being applied by Replicat. Normally, Oracle GoldenGate does not make these comparisons, because it is assumed that source and target data are identical before any given modification occurs. However, you cannot assume a synchronized state when the same row can be changed on either system. In this case, you need a before-after comparison to apply conflict resolution rules. To see before images in use, see the conflict-resolution example on page 96.

To use before values

1. To extract before values, use the GETUPDATEBEFORES parameter.
2. To reference a before value in a conflict-resolution procedure, use the following format:

```
BEFORE.<column_name>
```

Use fetched column values

If the database only logs modified columns to the transaction log, you might need to fetch column values from the database to use in your conflict-resolution routines, such as a TIMESTAMP column and foreign-key columns.

To fetch column values from the database

Use the FETCHCOLS and FETCHCOLSEXCEPT options of the TABLE statement in the Extract parameter file to fetch the values of the specified columns from the database.

(Oracle) To increase the accuracy of using before values from an Oracle database, Oracle GoldenGate recommends issuing the ADD TRANDATA command with the COLS clause, instead of using FETCHCOLS. The columns required for conflict resolution will be included in the supplemental logging.

Methods for resolving conflict

The following are some methods that are typically used in conflict-resolution routines. Different methods can be used in combination so that if one method fails, another method is applied. If you need assistance with constructing your routines, contact Oracle Support. For more information, go to http://support.oracle.com.

Timestamp priority

You can resolve conflicts based on a timestamp, where the record that was modified first (or in some cases, last) always takes priority. To use timestamp priority:

- Each record must contain a timestamp column that contains the date and time that the record was inserted or updated. You can use a database trigger or you can modify the application code to place the timestamp in the column.
- The timestamps on all databases must be identical, and all servers must be in the same time zone.

To detect and resolve conflicts based on a timestamp, first, try to apply the row normally. Compare the before image of the timestamp column in the replicated row to the current timestamp column in the database.

- If they match, there is no conflict.
- If they are different, compare the timestamp of the row in the database to the after image of the timestamp from the replicated row.

The row with the oldest timestamp (or newest timestamp, if that is your rule) takes priority.

Trusted source priority

You can resolve conflicts based on the source of the transaction. Data changes from the *trusted source* take priority over changes from other sources. This method of resolution is relatively easy to implement. The trusted source could be as simple as a server location or as complex as a hierarchical structure that is based on a priority that is assigned to the database user that executes the transaction.

Combined timestamp and trusted source priority

You can combine the use of timestamps and a trusted source to resolve conflicts when one or the other is not sufficient. This might be necessary when a local change and a replicated change, both to the same row, occur simultaneously so that you cannot rely on a timestamp. A secondary conflict-resolution routine could implement trusted-source resolution.

Delta value priority

If an application uses UPDATE statements to decrement quantities by a set amount, rules that simply give priority to one source or timestamp over another cannot accurately resolve conflicts in this case, because the net effect of both changes is not resolved. The data values on both systems will not be accurate. Cases like this include inventory balances, account balances, and sales balances. These cases must be resolved quantitatively by comparing a before image to the current image and then replicating the net change value instead of the actual values. See the example on page 97.

SQL-specific rules for replication priority

You can create rules for each type of SQL operation.

- Inserts: If the value of the key in a replicated insert matches the value of the key of an existing record in the target table, you can replace the existing row with the replicated one.
- Updates: If the key of a replicated update does not exist in the target table, you can transform an update into an insert.
- Deletes: If the key of a replicated delete does not exist in the target table, you can ignore the delete, assuming your business policies permit it. If a row is deleted from one database, it may not matter that the row does not exist in another database, because the record is supposed to be deleted anyway.

To keep track of the original and transformed values, use an exceptions table. See "Resolving conflicts through Oracle GoldenGate" on page 91.

Application-specific rules

You can use SQL procedures and user exits that are executed by Oracle GoldenGate to support built-in conflict-resolution methods that are supplied with applications that support distributed processing. Such methods include IP persistent routers or application privileges that prevent multiple users from modifying the same data. See Chapter 18 for information about using SQL procedures and user exits.

Terminate processing

You can terminate Replicat processing so that the error can be resolved before any other changes are made to the table(s). This is recommended only if the automatic conflict-resolution routine does not work properly or encounters an exception. Stopping Replicat increases the latency of the target data.

Handling resolution notification

When a conflict has been detected and resolved, the user of the business application might need to be notified that his or her expected outcome has changed. For example, when two customers reserve airline seats and both are assigned the same one, someone must be notified that an alternate seat has to be (or was) assigned.

The easiest way to handle notifications is to create a batch job that runs periodically on an exceptions table, issues the appropriate message, obtains the alternate results for the user, and makes any necessary updates to the database to reflect those changes. The job should be executed as a separate database user so that the work only affects data that has been committed to the database.

See "Resolving conflicts through Oracle GoldenGate" on page 91 for more information about using an exceptions table.

Conflict detection and resolution examples

Conflict resolution based on timestamp

This is an example of basic conflict detection based on a timestamp.

```
MAP swilkes.date_test, TARGET swilkes.date_test, &
REPERROR (21000, DISCARD), &
SQLEXEC (ID lookup, ON UPDATE, &
QUERY "select count(*) conflict from date_test where t_id = ? and &
t_timestamp > ?", &
PARAMS (p1 = t_id, p2 = t_timestamp), BEFOREFILTER, ERROR REPORT, &
TRACE ALL),&
FILTER (lookup.conflict = 0, ON UPDATE, RAISEERROR 21000);
```

NOTE The example does not show a complete Replicat parameter file, but shows only those relating to conflict resolution.

In this example, the goal is to prevent Replicat from applying a replicated UPDATE record if the existing target record is newer. The most recent change "wins."

The conflict detection code uses a SQLEXEC query to select all records in the target table that have the same key as, but a newer timestamp than, that of the replicated record. The affected columns are the t_id key column and the t_timestamp timestamp column. The query executes only on UPDATE operations, and does so before the FILTER clause so that the output can be applied to the filter rule. The query runs under the logical name of "lookup."

The result of the query is assigned to a variable named "conflict," which is used as input to the FILTER clause by using the lookup.conflict notation.

The FILTER clause states the following:

- If the result of the query is 0 (target record is older), the filter succeeds and the replicated record is applied to the target.
- If the result of the query is 1 (target record is newer), the filter fails and a user-defined error is raised by means of RAISEERROR. This error triggers the error handling response that is specified with REPERROR. In this example, the replicated record is discarded.

NOTE Multiple FILTER statements could be used to apply additional rules. They would be executed in the order that they appear in the parameter file.

This example also could be used with an exceptions MAP statement by:

- changing the REPERROR action to EXCEPTION instead of DISCARD.
- using the exceptions MAP statement to map the failed record (operation) to an exceptions table.

For example:

```
MAP swilkes.date_test, TARGET swilkes.date_test, &
REPERROR (21000, EXCEPTION), &
SQLEXEC (ID lookup, ON UPDATE, &
QUERY "select count(*) conflict from date_test where t_id = ? and &
t_timestamp > ?", &
PARAMS (p1 = t_id, p2 = t_timestamp), BEFOREFILTER, ERROR REPORT, &
TRACE ALL),&
FILTER (lookup.conflict = 0, ON UPDATE, RAISEERROR 21000);

MAP swilkes.date_test, TARGET swilkes.date_test_exc, EXCEPTIONSONLY, &
INSERTALLRECORDS, &
COLMAP (USEDEFAULTS, errtype = "UPDATE FILTER FAILED.");
```

The second MAP statement is the exceptions MAP statement. The EXCEPTIONSONLY option causes the exceptions MAP statement to become active only when an error occurs for the last record that was processed by the preceding MAP statement (which maps the same swilkes.date_test source table). The INSERTALLRECORDS parameter causes Replicat to insert each operation that causes an error as a new record in the swilkes.date_test_exc exceptions table. This preserves all conflict history as point-in-time snapshots.

An exceptions table such as this one would typically contain the same columns as the source table, plus additional columns that capture contextual information. In this example, the extra column "errtype" is being used to reflect the cause of the error (failed filter). For help with constructing an exceptions table, see page 92.

Conflict-resolution based on net change values

This is an example of how to maintain accurate inventory quantities on both systems when the application uses UPDATE statements to decrement column values. Instead of replicating actual values from the updates, it compares the before image with the after image and replicates the delta instead.

In this example, a "Bowl" item in inventory is at a quantity of 10. Two customers, logging into different databases, who order the same item should both have successful orders if one purchases 3 units and the other purchases 5 units, leaving a total of 2 items in the physical inventory.

The update to the inventory on each system is:

SystemA (3 bowls purchased):

```
Update inventory set quantity = 7 where item = 'Bowl' and quantity = 10;
```

SystemB (5 bowls purchased):

```
Update inventory set quantity = 5 where item = 'Bowl' and quantity = 10;
```

When the transaction from SystemA is applied to SystemB, it fails because the before image of the quantity on System B is expected to be 10, but instead it is 5. Conversely, when the transaction from SystemB is applied to SystemA, it fails because the before image is expected to be 10, but instead it is 7.

To resolve this conflict, calculate the net change in inventory at the source of the change and then apply that to the target table in the replicated transaction, rather than using the actual numbers. This is the formula:

```
Update inventory set quantity = (before image - after image) where item
= ‘Bowl’;
```

Apply to SystemA:

```
Update inventory set quantity = (5 - 3) where item = ‘Bowl’;
```

Apply to SystemB:

```
Update inventory set quantity = (7 - 5) where item = ‘Bowl’;
```

After both of those statements succeed, the quantity of bowls remaining is 2 in both locations, the correct amount.

CHAPTER 9

Configuring Oracle GoldenGate security

Overview of security options

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

Table 4 Oracle GoldenGate security options

Security feature	Description
Encryption	Options are available for encrypting and decrypting: ◆ data in an extract file or trail ◆ database passwords ◆ data sent across TCP/IP
Command security	Sets user-level permissions for accessing Oracle GoldenGate commands through GGSCI.
Connection security	Allows connections to be established from the target system instead of the source system. For use when the target resides within a trusted network zone behind an internal firewall.

Using encryption

This section contains instructions for encrypting and decrypting the following:

- The trail or extract file that holds data being processed by Oracle GoldenGate
- A database password
- The data sent across TCP/IP

How data is encrypted

The following encryption methods are used:

- To encrypt trail or extract files, Oracle GoldenGate uses 256-key byte substitution. All records going into those files are encrypted both across any data links and within the files themselves.

- To encrypt the database password or data that is sent across TCP/IP, Oracle GoldenGate uses Blowfish encryption. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. Oracle GoldenGate's implementation of Blowfish can take a variable-length key from 32 bits to 128 bits. Blowfish encryption can be combined with Oracle GoldenGate trail encryption.

Encrypting trail or extract files

You can encrypt the data in any local or remote trail or file.

NOTE (DB2 on z/OS) This feature 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.

To encrypt trail or extract files

1. In the Extract parameter file, list the following parameter before all trails or files that you want to be encrypted. You can list multiple trails or files after one instance of this parameter.

   ```
   ENCRYPTTRAIL
   ```

2. To disable encryption for any files or trails listed in the Extract parameter file, precede their entries with the following parameter.

   ```
   NOENCRYPTTRAIL
   ```

3. In the Replicat parameter file, include the following parameter so that Replicat decrypts the data for processing.

   ```
   DECRYPTTRAIL
   ```

   You also can use DECRYPTTRAIL for an Extract data pump to decrypt the data for column mapping, filtering, transformation, and so forth. You can then leave it decrypted for downstream trails or files, or you can use ENCRYPTTRAIL to encrypt the data again before it is written to those files.

Encrypting the password of a database user

You can encrypt any of the following database passwords through Oracle GoldenGate:

- The database password that is used by the Extract and Replicat processes and other processes to log into the source and target databases. (Not all database types require a database login for Oracle GoldenGate processes.)
- The database password for an Oracle ASM user.

To encrypt a database user password

1. Run GGSCI and issue the ENCRYPT PASSWORD command to generate an encrypted password. The command provides the following options.

   - The default ENCRYPT PASSWORD command, without any options, generates an encrypted password using a default key that is randomly generated by Oracle GoldenGate.

     ```
     ENCRYPT PASSWORD <password>
     ```

- ENCRYPT PASSWORD with the ENCRYPTKEY <keyname> option generates an encrypted password using a user-defined key contained in the ENCKEYS lookup file.

  ```
  ENCRYPT PASSWORD <password> ENCRYPTKEY <keyname>
  ```

  For <keyname>, specify the logical name for the key you want to use, as it appears in the local ENCKEYS file. To use this option, you must first generate a key, create an ENCKEYS file on the local system, and create an entry in the file for the generated key. For instructions, see "Generating encryption keys" on page 102.

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

2. Copy the encrypted password and paste it into the appropriate Oracle GoldenGate parameter statement as shown in Table 5.

   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.
   - <encrypted_password> is the encrypted password that is copied from the ENCRYPT PASSWORD command results.
   - ENCRYPTKEY DEFAULT is required if the password was encrypted using ENCRYPT PASSWORD without the ENCRYPTKEY option.
   - ENCRYPTKEY <keyname> is required if the password was encrypted using ENCRYPT PASSWORD with the ENCRYPTKEY <keyname> option. Specify the logical name of the key as it appears in the ENCKEYS lookup file.

Table 5 Specifying encrypted passwords in the Oracle GoldenGate parameter file

To encrypt a password for...	Use this parameter...
Oracle GoldenGate database user	`USERID <user>, PASSWORD <encrypted_password>, & ENCRYPTKEY {DEFAULT | <keyname>}`
Oracle GoldenGate user in Oracle ASM instance	`TRANLOGOPTIONS ASMUSER SYS@<ASM_instance_name>, & ASMPASSWORD <encrypted_password>, & ENCRYPTKEY {DEFAULT | <keyname>}`

Encrypting data sent across TCP/IP

You can encrypt captured data before Oracle GoldenGate sends it across the TCP/IP network to the target system. On the target system, Oracle GoldenGate decrypts the data before writing it to the Oracle GoldenGate trails (unless trail encryption also is specified). By default, data sent across a network is not encrypted.

To encrypt data sent across TCP/IP

1. On the source system, generate one or more encryption keys and create an ENCKEYS file. See "Generating encryption keys" on page 102.

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 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.
   ```

3. Depending on whether this is a regular Extract group or a passive Extract group (see page 106), use the ENCRYPT option of either the RMTHOST or RMTHOSTOPTIONS parameter to specify the type of encryption and the logical key name as shown:

   ```
   ENCRYPT BLOWFISH, KEYNAME <keyname>
   ```

   Where:

   ❍ BLOWFISH specifies Blowfish encryption.
   ❍ <keyname> is the logical name for the encryption key you want to use, as it appears in the ENCKEYS file.

   Examples:

   ```
   RMTHOST sys1, MGRPORT 7840, ENCRYPT BLOWFISH, KEYNAME superkey
   RMTHOSTOPTIONS ENCRYPT BLOWFISH, KEYNAME superkey
   ```

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

   ```
   -KEYNAME <name>
   -ENCRYPT BLOWFISH
   ```

   Where:

   ❍ KEYNAME <name> specifies the name of the key.
   ❍ ENCRYPT BLOWFISH specifies Blowfish encryption.

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

Generating encryption keys

You must create at least one encryption key and two ENCKEYS lookup files, one on the source and one on the target, if you want to:

❍ Encrypt data sent across TCP/IP
❍ Use a *user-defined* key to encrypt the database password

This procedure is *not required* if:

- you are using a default key generated by Oracle GoldenGate to encrypt the database password
- you are encrypting a trail or extract file.

You can define your own key or run Oracle GoldenGate's KEYGEN utility to create a key randomly.

To define your own key

- The key name can be a string of 1 to 24 alphanumeric characters without spaces or quotes.
- The key value can be up to 128 bits (16 bytes) as a quoted alphanumeric string (for example "Dailykey") or 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.

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

Where:

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

Example:

```
KEYGEN 128 4
```

To store the keys for use by Oracle GoldenGate

1. On the source system, open a new ASCII text file.
2. For each key that you generated, enter a logical name followed by the key value itself. 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 ENCKEYS without an extension in the Oracle GoldenGate installation directory. The name must be in upper case.
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.
```

Using 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 105, 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 6 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 7 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 7 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

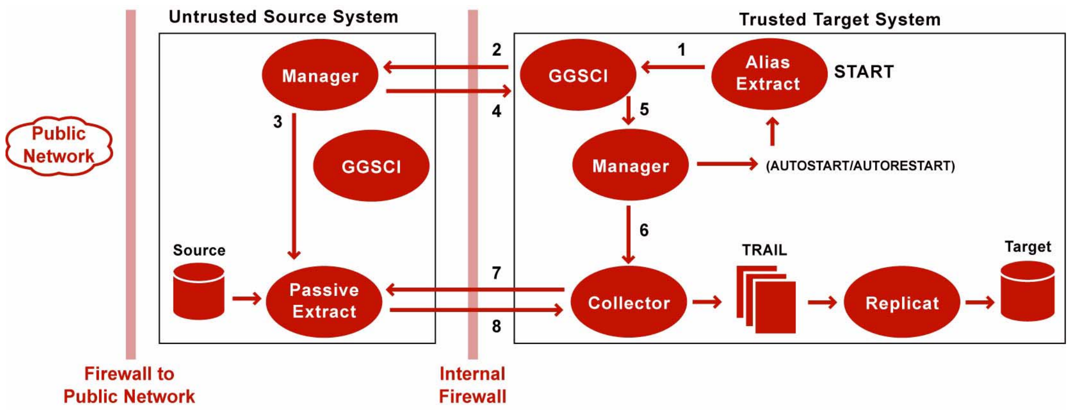
Because the CMDSEC file is a source of security, it must be secured. You can grant read access as needed, but Oracle GoldenGate recommends denying write and delete access to everyone but Oracle GoldenGate Administrators.

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 finds an open port (according to rules in the DYNAMICPORTLIST Manager parameter) and starts the passive Extract, which listens on the specified 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.

```
RMTHOSTOPTIONS
[, COMPRESS]
[, COMPRESSTHRESHOLD]
[, ENCRYPT {NONE | BLOWFISH}]
[, KEYNAME <keyname>]
[, PARAMS <collector parameters>]
[, TCPBUFSIZE <bytes>]
[, TCPFLUSHBYTES <bytes>]
```

Instructions for using the ENCRYPT and KEYNAME options begin on page 101 of this chapter. These options encrypt data sent across TCP/IP using Blowfish encryption. For information about the rest of the RMTHOSTOPTIONS options, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

For more information about configuring an Extract group, see Chapter 12.

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 113.

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).
- This configuration can be used when Extract is started from the command line for a batch run. See Chapter 13 on page 135.
- 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 10
Handling Oracle GoldenGate processing errors

Overview of Oracle GoldenGate error handling

Oracle GoldenGate provides error-handling options for:

- Extract
- Replicat
- TCP/IP

Handling Extract errors

There is no specific parameter to handle Extract errors when DML operations are being extracted, but Extract does provide a number of parameters that can be used to prevent anticipated problems. These parameters handle anomalies that can occur during the processing of DML operations, such as what to do when a row to be fetched cannot be located, or what to do when the transaction log is not available. The following is a partial list of these parameters. For a complete list, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

- FETCHOPTIONS
- WARNLONGTRANS
- DBOPTIONS
- TRANLOGOPTIONS
- LFMMEMORY

To handle extraction errors that relate to DDL operations, use the DDLERROR parameter. For more information, see page 173.

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.

All options but TRANSDISCARD and TRANSEXCEPTION affect only the individual record that generated an error. TRANSDISCARD and TRANSEXCEPTION 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.)

The following comprise the full 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.

Handling errors as exceptions

You can treat Replicat errors as exceptions by using the REPERROR parameter with the EXCEPTION option or the TRANSEXCEPTION option. You can then map those exceptions to an exceptions table, along with information about the error that can be used to resolve the error. You can use the following options of the MAP statement to send the exceptions to the exceptions table:

- EXCEPTIONSONLY (requires a separate exceptions MAP statement)
- MAPEXCEPTION (requires only one MAP statement)

Using EXCEPTIONSONLY

You can use the EXCEPTIONSONLY clause if the names of the source and target tables will *not* be wildcarded within the MAP and TARGET clauses of the MAP statement. With this method, you must create two MAP statements for a source table:

- The first, the regular 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). The exceptions MAP statement executes only after an error on the source table. The exceptions MAP statement must immediately follow the regular MAP statement that contains that source table.

Using MAPEXCEPTION

You must use MAPEXCEPTION if the names of the source and target tables *will* be wildcarded within the MAP and TARGET clauses of the MAP statement. You can include the MAPEXCEPTION clause in the regular MAP statement, the same one in which you map the source tables to the target tables. The MAPEXCEPTION clause captures the operations that are treated as exceptions, based on the REPERROR statement, and maps them to the exceptions table. In this configuration, only one MAP statement is needed instead of two.

About the exceptions table

The exceptions table provides information about an error that you can use in any manner you wish, such as configuring your application to handle the error or initiate conflict resolution. An exceptions table typically is created with the same columns as the regular target table, plus extra columns to capture additional information about the data and the error. You can use COLMAP, Oracle GoldenGate conversion functions, and other applicable Oracle GoldenGate options to get this information, such as getting the error number and environment information with a function and mapping the information to the appropriate column with COLMAP.

Example 1 EXCEPTIONSONLY

This is an example of how to use EXCEPTIONS only for exceptions mapping. It shows an example of how to use REPERROR with EXCEPTIONSONLY and an exceptions MAP statement. This example only shows those parameters that relate to REPERROR; others are omitted to save space in this documentation.

```
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.

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 Extract DDL processing errors” on page 173.

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 17 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 8:

Table 8 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 11
Creating a data-definitions file

Overview of the data-definitions file

A data-definitions file contains information about the format of data that is being replicated, such as the table name, column names, data types, data length, and offset. A data-definitions file enables Oracle GoldenGate to convert data from one format to another when moving data between different kinds of databases. To perform conversions, the definitions of both sets of data must be known to Oracle GoldenGate. An Oracle GoldenGate process can query the database that is local to that process, in order to get one set of definitions, but that process must rely on a definitions file to get the set of definitions for the remote database. For example, the Replicat process can query the target database, but relies on a definitions file to get metadata for the source database.

When to use a data-definitions file

A data-definitions file is required whenever you are synchronizing source and target tables that have dissimilar data definitions, for example Oracle source tables and Microsoft SQL Server target tables. Even when the source and target database systems are identical, the source and target tables can be dissimilar when corresponding source and target columns *do not* meet the guidelines in "Rules for tables to be considered identical".

Rules for tables to be considered identical

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

- have identical column names (including case, if applicable)
- have identical data types
- have identical column sizes
- have the same column length semantics for character columns (bytes versus characters)
- appear in the same order in each table

For example, a source-definitions file is required when the semantics of a source Oracle database are configured as bytes and the target semantics are configured as characters (or the other way around), even though the table structures may be identical. As another example, a source-definitions file is required for the following set of source and target tables, which are identical except for the order of the name columns:

Source	**Target**

`CREATE TABLE emp`
`( employee_id    NUMBER(6)`
`, first_name     VARCHAR2(20)`
`, last_name      VARCHAR2(25)`
`, phone_number   VARCHAR2(20)`
`, hire_date      DATE  DEFAULT SYSDATE`
	`CREATE TABLE emp`
	`( employee_id    NUMBER(6)`
	`, last_name      VARCHAR2(25)`
	`, first_name     VARCHAR2(20)`
	`, phone_number   VARCHAR2(20)`
	`, hire_date      DATE  DEFAULT SYSDATE`

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

Types of definitions files

- If you are configuring Oracle GoldenGate to perform column mapping or transformation on the target system, a *source-definitions file* is required. The source-definitions file contains the definitions of the source tables. You transfer this file to the target system. Replicat refers to those definitions, plus the queried target definitions, to perform the conversions.
- If you are configuring Oracle GoldenGate to perform column mapping or transformation on the source system, a *target-definitions file* is required. The target-definitions file contains the definitions of the target tables. You transfer the file to the source system. A primary Extract or a data-pump refers to those definitions, plus the queried source definitions, to perform the conversions.
- When replicating to a NonStop Server target from any other type of database, the mapping and conversion must be performed on the source Windows or UNIX system. Replicat for NonStop does not convert two-part table names and data types to the three-part names that are used on the NonStop platform, so Extract must format the trail data with the NonStop names and target data types. Therefore, a data definitions file from the target is always required in this situation to support the conversion.
- If you are configuring Oracle GoldenGate to perform column mapping or transformation on an intermediary system that contains *neither* a source database nor a target database, you must provide a source-definitions file *and* a target-definitions file on that system.

When to use a definitions template

If you think that you will be adding tables to the Oracle GoldenGate environment after the initial configuration and startup, use a definitions template. You can generate a definitions template that works for multiple tables if those tables all have exactly the same structure. For example, a template is appropriate if you have separate tables for each customer, and all of the columns, the column order, and the data types in each one are identical.

By using templates, you eliminate the need to generate a new definitions file for each new table, and you avoid having to stop and start Oracle GoldenGate processes to activate the new definitions. This reduces your maintenance work when new tables are added frequently.

Templates work like this:

- You specify the definitions template when you create the master data-definitions file.
- Whenever you add a new table that has the same structure as one that is in the Oracle GoldenGate configuration already, you can map the new table to the template with the DEF or TARGETDEF option of the TABLE or MAP parameter.

NOTE If you do not use a template in the master definitions file, you can always generate a definitions file for each new table that you add to the Oracle GoldenGate configuration. You can simply copy the contents of each new definitions file and add them to the existing master definitions file.

Configuring a data-definitions file

To configure Oracle GoldenGate to use a data-definitions file, you:

- create a parameter file for the DEFGEN utility
- run the DEFGEN utility to generate the file
- point the Oracle GoldenGate process to the definitions file

To create a parameter file for DEFGEN

To create a source-definitions file, perform these steps on the source system. To create a target-definitions file, perform the steps on the target system.

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

Table 9 DEFGEN parameters

Parameter	Description
`DEFSFILE <full_pathname>`	Specifies the relative or fully qualified name of the data-definitions file that is to be the output of DEFGEN.

`[{SOURCEDB | TARGETDB} <dsn>,]` `[USERID <user>[, PASSWORD <password>]]`
 ◆ SOURCEDB|TARGETDB specifies a data source name, if required as part of the connection information. Not required for Oracle.
 ◆ USERID and PASSWORD are required if database authentication must be provided. A password is not required for SQL/MX or DB2. For Oracle, you can include a host string, for example:
 `USERID ggs@ora1.ora, PASSWORD ggs123`
	Specifies database connection information.
	 For more information about SOURCEDB and USERID, see the Oracle GoldenGate *Windows and UNIX Reference Guide.*

Table 9 **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 wildcards.
◆ [, {DEF | TARGETDEF} <template name>] specifies that the definitions of this table are being used to create the specified template. This option is not supported for initial loads.
	Specifies a table or tables for which definitions will be defined. Optionally, specifies a table on which to base a definitions template.
	Specify a source table or tables if generating a source-definitions file for transfer to the target, or specify a target table or tables if generating a target-definitions file that will be transferred to a source system.
	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.

To 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]
```

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.

NOTE Do not modify the definitions file.

2. Using ASCII mode, FTP the definitions file from the Oracle GoldenGate dirdef sub-directory to the other system, saving it to the dirdef sub-directory on that system.

To point a Oracle GoldenGate process to a definitions file

Link a data-definitions file to the correct Oracle GoldenGate process in the following ways:

❍ Link a target-definitions file to an Extract group or data pump by using the TARGETDEFS parameter in the Extract parameter file.
❍ Link a source-definitions file to 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, link a source-definitions file and a target-definitions file to 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.

Example The following is an example of a DEFGEN parameter file for an Oracle database. Individual definitions by name will be created for all tables in the ord and hr schemas. In addition, a custdef template is being created based on table acct.cust100. In the database, there are other acct.cust* tables, each with identical definitions to acct.cust100. Those tables can be mapped to a DEF clause in the TABLE or MAP statement that points to the custdef template.

```
DEFSFILE C:\ggs\dirdef\record.def
USERID ggs, PASSWORD ggs
TABLE acct.cust100, DEF custdef;
TABLE ord.*;
TABLE hr.*;
```

The tables would be mapped in the Replicat parameter file as follows:

```
REPLICAT acctrep
SOURCEDEFS c:\ggs\dirdef\record.def
USERID ggs, PASSWORD ggs
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;
```

In the previous MAP statements, definitions for all tables matching the wildcard specification acct.cust* are obtained from the custdef template.

The same tables could be mapped for a primary Extract or a data pump, if target definitions are required as well as source definitions. For example, if the target was an Enscribe database, or if the mapping or conversion is being done on an intermediary system, the configuration would look similar to the following:

```
EXTRACT acctext
USERID ggs, PASSWORD ggs
RMTHOST sysb, MGRPORT 7890
RMTTRAIL $data.ggsdat.rt
TABLE acct.cust*, TARGET acct.cust*, DEF custdef, TARGETDEF tcustdef;
TABLE ord.prod, TARGET ord.prod;
TABLE ord.parts, TARGET ord.parts;
TABLE hr.emp, TARGET hr.emp;
TABLE hr.salary, TARGET hr.salary;
```

In the previous example, a source template named custdef and a target template named tcustdef will be used for all acc.cust* tables. Definitions for the tables from the ord and hr schemas will be obtained from explicit definitions based on the table names.

CHAPTER 12
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 214 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.
- Do not place a numeric value at the end of a group name, such as fin1, fin10, and so forth. You can place a numeric value at the beginning of a group name, such as 1_fin, 1fin, and so forth.

Creating a checkpoint table

NOTE The checkpoint table feature is not supported for c-tree databases, because Replicat does not have access to the target database.

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 16.

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>]]
   ```

Where:

- SOURCEDB <dsn> supplies a data source name, if required as part of the connection information.
- USERID <db_user>, PASSWORD <pw> supplies database credentials, 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 121 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).
  - `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 106 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 14.

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 10).

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 50
```

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 106.

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

Table 10 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>]]`
◆ 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 ggs123`
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*.
	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.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
	Specifies the target system and port where Manager is running. 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.

Table 10 Online change-extraction parameters (continued)

Parameter	Description
`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).
`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")`

Table 10 Online change-extraction parameters (continued)

Parameter	Description
`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 121 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 120).
- ❍ 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 11 in the order shown, starting a new line for each parameter statement.

Table 11 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.

Table 11 Online change-replication parameters (continued)

Parameter	Description
`{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 11 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.

`[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>]]`
◆ 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 ggs123`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide.*

Table 11 Online change-replication parameters (continued)

Parameter	Description

`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 11.)
	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 16 on page 214.

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 106 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 106 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>]
   ```

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>]]
   ```

   Where:

   ❍ SOURCEDB <dsn> supplies the data source name, if required as part of the connection information.

   ❍ USERID <user>, PASSWORD <password> supplies database credentials, 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 13
Configuring change synchronization as a batch run

Overview of batch change synchronization

You can configure Extract and Replicat to perform an individual batch run (or *special run*) to extract and replicate data changes that were generated between a specific start time and end time. Checkpoints are not recorded during a batch run, because a recovery point is not needed. In the event of a process failure, you can simply start the run over again using the same start and end points.

The following summarizes the steps required to establish a batch run:

- Create a batch Extract parameter file.
- Create a batch Replicat parameter file.
- Start processes from the command shell of the operating system.

If this is the first time you are running Oracle GoldenGate to synchronize data changes, you may need to perform an initial load to prepare the target tables for synchronization. An initial load takes a copy of entire source tables, transforms the data if necessary, and applies it to the target tables so that the movement of transaction data begins from a synchronized state. See "Running an initial data load" on page 214.

Creating a parameter file for batch extraction

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

   ```
   EDIT PARAMS <name>
   ```

   Where: <name> is a name for the Extract group, up to eight characters, not case-sensitive.
3. Enter the parameters listed in Table 12 in the order shown, starting a new line for each parameter statement.

Table 12 Batch change-extraction parameters

Parameter	Description

`SPECIALRUN`
`TRANLOG [<bsds name>] |`
`EXTTRAILSOURCE <trail name>] |`
`EXTFILESOURCE <file name>}`
◆ TRANLOG extracts from the transaction logs. Use for log-based extraction. Use the <bsds> option for DB2 on z/OS to specify the Bootstrap Data Set file name of the transaction log.
◆ EXTTRAILSOURCE <trail name> extracts from a local trail. Use when processing from a set of trails.
◆ EXTFILESOURCE extracts from a local extract file. Use for a data pump.
Specify relative or full path names for both file types.
	Configures Extract as a batch run for which checkpoints are not required.
	If using a data pump, you can extract either from a trail or an extract file.

`BEGIN <begin time>` ◆ <begin time> is a date in the format of yyyy-mm-dd hh:mi[:ss[.ccccccc]].	Specifies the transaction commit time at which to start processing.
`END {<end time> | RUNTIME}` ◆ <end time> is a date in the format of yyyy-mm-dd [hh:mi[:ss[.cccccc]]. ◆ RUNTIME causes Extract to terminate when it reaches process startup time. With RUNTIME, you need not alter the parameter file to change dates and times from run to run.	Specifies the transaction commit time at which to stop processing.

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
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*.
	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.

`RMTHOST <hostname>,`
`MGRPORT <portnumber>`
	Specifies the target system and port where Manager is running. This option is only required when sending data over IP to a remote system.

Table 12 Batch change-extraction parameters (continued)

Parameter	Description

`RMTFILE <full_pathname> |`
`EXTFILE <full_pathname>`
◆ RMTFILE specifies an extract file on the target system.
◆ EXTFILE specifies a local extract file (for use with a data pump).
Specify a relative or fully qualified file name.
	Specifies an extract file in which data changes will be stored temporarily. If specifying multiple files, follow each designation with the appropriate TABLE statements.
	If 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).

`PASSTHRU | NOPASSTHU`	(For a data pump) Specifies whether or not subsequent TABLE specifications will use normal or pass-through processing.
`TABLE <owner>.<table>;` ◆ <owner> is the schema name. ◆ <table> is the name of the table or a group of tables defined with wildcards.	Specifies a table or tables for which to extract data changes. To exclude tables from a wildcard specification, use the TABLEEXCLUDE parameter.

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

Creating a parameter file for batch replication

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

```
EDIT PARAMS <name>
```

Where: <name> is a name for the Replicat group, up to eight characters, not case-sensitive.

3. Enter the parameters listed in Table 13 in the order shown, starting a new line for each parameter statement.

Table 13 Batch change-replication parameters

Parameter	Description
`SPECIALRUN`	Configures Replicat as a batch run.
`BEGIN <begin time>` ◆ <begin time> is a date in the format of yyyy-mm-dd hh:mi[:ss[.cccccc]].	Specifies the transaction commit time at which to start processing.
`END {<end time> | RUNTIME}` ◆ <end time> is a date in the format of yyyy-mm-dd [hh:mi[:ss[.cccccc]]. ◆ RUNTIME causes Replicat to terminate when it reaches process startup time. With RUNTIME, you need not alter the parameter file to change dates and times from run to run.	Specifies the transaction commit time at which to stop processing.
`EXTFILE <file name>`	Specifies the relative or fully qualified name of the extract file that was specified with RMTFILE or EXTFILE in the Extract parameter file.
`{SOURCEDEFS <file name>} | ASSUMETARGETDEFS` ◆ Use SOURCEDEFS if the source and target tables have different definitions. Specify the relative or full path name of the source-definitions file generated by DEFGEN. See Chapter 11 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.

`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 the file to which Replicat writes rejected record data, for example records that generated database errors. A discard file is optional but recommended.

Table 13 Batch change-replication parameters (continued)

Parameter	Description

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. See the Oracle GoldenGate *Windows and UNIX Reference Guide*.

`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 11.)
	Specifies a relationship between a source and target table or tables.
	To exclude tables from a wildcard specification, use the MAPEXCLUDE parameter.

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

Starting processes from the command shell of the operating system

To start a batch change-synchronization job, run the extract and replicat programs from the command shell of the operating system. A batch run terminates by itself according to the END parameter.

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 processes from the command shell

1. (Valid only for a passive-alias Extract configuration) Start a static Collector process.

```
server -h <host> -p <port>
```

Where:

❍ -h <host> is the name or IP address of the source system.
❍ -p <port is the port number where Extract on that system will be listening for Collector to open a connection.

2. From the Oracle GoldenGate directory on the source system, run GGSCI.

3. In GGSCI on the source and target systems, start Manager.

   ```
   START MANAGER
   ```

   NOTE In a Windows cluster, start the Manager resource from the Cluster Administrator.

4. On the source and target systems, issue one of the following sets of commands, depending on the process you are starting. Run the programs from the Oracle GoldenGate directory.

   ```
   extract paramfile <name>.prm reportfile <name>.rpt [-p <port>]
   ```

   or...

   ```
   replicat paramfile <name>.prm reportfile <name>.rpt
   ```

   Where:

   - paramfile <name>.prm is the relative or fully qualified name of the parameter file. The command name can be abbreviated to pf.
   - reportfile <name>.rpt is the relative or fully qualified name of the report file. The command name can be abbreviated to rf.
   - -p <port> is the local port number where Extract will be listening for Collector to open a connection. Use this option only to start Extract in passive mode. For more information on the passive-alias Extract configuration, see page 106.

     NOTE In batch mode, an alias Extract is not required because GGSCI is not being used to start processes.

CHAPTER 14
Configuring DDL synchronization for an Oracle database

Overview of DDL synchronization

Oracle GoldenGate supports the synchronization of DDL operations from one database to another. DDL synchronization can be active when:

- business applications are actively accessing and updating the source and target objects.
- Oracle GoldenGate transactional data synchronization is active.

The components that support the replication of DDL and the replication of transactional data changes (DML) are independent of each other. Therefore, you can synchronize:

- just DDL changes
- just DML changes
- both DDL and DML

For example, if you use batch runs to keep the target objects current, you can configure DDL synchronization as a continuous (online) run, so that the target metadata is always up-to-date when the batch loads are performed. The Oracle GoldenGate batch load will use the current metadata from the source and target catalogs.

For a list of supported objects and operations for DDL support for Oracle, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.

Limitations of Oracle GoldenGate DDL support

DDL statement length

Oracle GoldenGate measures the length of a DDL statement in bytes, not in characters. The supported length is approximately 2 MB, allowing for some internal overhead that can vary in size depending on the name of the affected object and its DDL type, among other characteristics. If the DDL is longer than the supported size, Extract will issue a warning and ignore the DDL operation.

The ignored DDL is saved in the marker table. You can capture Oracle DDL statements that are ignored, as well as any other Oracle DDL statement, by using the ddl_ddl2file.sql script, which saves the DDL operation to a text file in the USER_DUMP_DEST directory of

Oracle. The script prompts for the following input:

- The name of the schema that contains the Oracle GoldenGate DDL objects, which is specified in the GLOBALS file
- The Oracle GoldenGate marker sequence number, which is recorded in the Extract report file when DDLOPTIONS with the REPORT option is used in the Extract parameter file
- A name for the output file

System configuration

- Oracle GoldenGate supports DDL replication in uni-directional configurations and also in bi-directional configurations (active-passive and active-active) between two, and only two, systems. For special considerations in an Oracle active-active configuration, see “Propagating DDL in an active-active (bi-directional) configurations” on page 168.
- Oracle GoldenGate supports DDL synchronization only in a like-to-like configuration. Oracle GoldenGate DDL support requires the following:
    - Source and target object definitions must be identical.
    - The ASSUMETARGETDEFS parameter must be used in the Replicat parameter file. Replicat will abend if objects are configured for DDL support and the SOURCEDEFS parameter is being used. For more information about ASSUMETARGETDEFS, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Filtering, mapping, and transformation

DDL

DDL operations cannot be transformed by any Oracle GoldenGate process. You can use simple string substitution, and you can map schema and object names as follows:

- Source DDL can be mapped and filtered to a different target object by a primary Extract or a Replicat process, but the mapping or filtering of DDL by a data-pump Extract is not permitted. DDL is propagated through a data pump in PASSTHRU mode (non-mapped or filtered).
- As a result, DDL that is performed on a source table of a certain name (for example ALTER TABLE TableA...) will be processed by the data pump with the same table name (ALTER TABLE TableA). It cannot be mapped by that process as ALTER TABLE TableB, regardless of any TA BLE statements that specify otherwise.

DML

Because DDL is passed through unchanged and unmapped by a data pump , consider this limitation if you intend to perform DML manipulation that maps source tables to different target names. Perform filtering, mapping, and transformation of DML with the primary Extract or with Replicat, and configure those tables for PASSTHRU mode in the data pump .

If a data pump must performDML manipulation that includes name mapping, then you must also configure Replicat to perform the corresponding name mapping for replicated DDL for those tables.

NOTE Tables that do not use DDL support can be configured in NOPASSTHRU mode to allow data filtering, and manipulation by the data pump.

To configure a table for data pass-through

1. In the parameter file of the data pump , place the PASSTHRU parameter before all of the TABLE statements that contain tables that use DDL support.
2. In the same parameter file, you can place the NOPASSTHRU parameter before any TABLE statements that contain tables that do not use DDL support, if you want data filtering, mapping, or transformation to be performed for them.
3. Do not use any of the DDL configuration parameters for a data pump: DDL, DDLOPTIONS, DDLSUBST, PURGEDDLHISTORY, PURGEMARKERHISTORY, DDLERROR, or any of the Oracle GoldenGate tracing parameters with DDL options.

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

SQLEXEC

- 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.

User Exits

Use the GET_DDL_RECORD_PROPERTIES function to return a DDL operation, including information about the object on which the DDL was performed and also the text of the DDL statement itself. The Extract process can only get the source table layout. The Replicat process can get source or target layouts.

This user exit only provides retrieval capability. Oracle GoldenGate does not provide a function for manipulating DDL records.

Special DDL cases and their treatment

Truncates

TRUNCATE statements can be replicated as follows:

- As part of the Oracle GoldenGate full DDL support, which supports TRUNCATE TABLE, ALTER TABLE TRUNCATE PARTITION, and other DDL as documented in the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*
- As standalone TRUNCATE support. This support enables you to replicate TRUNCATE TABLE, but no other DDL. The GETTRUNCATES parameter controls the standalone TRUNCATE feature. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

To avoid errors from duplicate operations, only one of these features can be active at the same time.

Renames

- RENAME operations on tables are converted to the equivalent ALTER TABLE RENAME. For example RENAME tab1 TO tab2 would be changed to ALTER TABLE tab1 RENAME TO tab2. The reason for this conversion is that RENAME does not support the use of a schema name, but ALTER TABLE RENAME does. Oracle GoldenGate makes the conversion so that a schema name can be included in the target DDL statement. The conversion is reported in the Replicat process report file.
- An ALTER TABLE RENAME will fail if the old or new table name is longer than 18 characters (16 for the name and two for the quotation marks). Oracle only allows 18 characters for a rename because of the ANSI limit for identifiers.
- RENAME operations on sequences and views cannot be converted to an ALTER statement, because there is no such statement in Oracle for sequences and views. Consequently, sequence renames are always replicated on the target with the same owner and object name as in the source DDL and cannot be mapped to something different.

LOB columns

It is possible for DDL to be executed on a source object between the time when a data (DML) operation occurs on that object and the time when Extract captures that operation from the redo log, if there is Extract lag. Because Extract processes transaction records sequentially, and because both the DDL and DML are recorded in the log, the new metadata usually is resolved before the DML record is encountered. However, in the case of a LOB, Extract might need to fetch a LOB value from a Flashback Query, which can provide metadata out of sequence.

The reason for this inconsistency is that Oracle does not provide Flashback capability for DDL (except DROP). When a LOB is fetched, the object structure reflects current metadata, but the LOB record in the transaction log reflects old metadata.

To resolve these differences in structure, Oracle GoldenGate compiles a common set of columns that match in name, type and length, and then the LOB data is fetched from those columns. The net result is:

- Fetched data could be newer than the data that is being processed by Extract or, in the case of columns that were dropped, not present.
- The DDL could drop a column and then recreate it with the same name but a different data type. (This is the worst-case scenario). In this case, the incompatibility between the transaction record (old data type) and the database record (new data type) can cause Replicat processing errors.

To prevent LOB inconsistencies

- Keep Extract lag small or perform DDL operations on tables that contain LOBs when transactional volume is low or absent, and only after the DML in the Oracle GoldenGate trail is processed by Replicat. You can find suggestions for reducing lag in the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.
- If there are DML operations remaining to be processed by Replicat, do not execute DDL on the columns that are being used by Oracle GoldenGate as row identifiers on tables that contain LOBs. A row identifier can be the primary key columns, columns on which

a unique key is defined, or columns configured as a substitute key with a KEYCOLS clause in a TABLE or MAP parameter. In the absence of any of those identifiers, the row identifier will be all of the columns in the row. If DDL must be performed on identifier columns, take the following steps.

To perform DDL on row identifiers in tables that contain LOBs

1. Pause source DML operations.
2. Wait until Replicat finishes processing all of the data in the trail. To determine when Replicat is finished, issue the following command until you see a message that there is no more data to process.

   ```
   INFO REPLICAT <group>
   ```

3. Execute the DDL on the source.
4. Resume source DML operations.

User defined types

- DDL operations that involve user defined types generate implied DML operations on both the source and target. To avoid SQL errors that would be caused by redundant operations, Oracle GoldenGate does not replicate those DML operations.
- If DML is being replicated for a user defined type, Extract must process all of those changes before DDL can be performed on the object. Because UDT data might be fetched by Extract, the reasons for this rule are similar to those that apply to LOB columns. (See the "LOB columns" topic.)

To perform DDL on an Oracle UDT if change data is being captured

1. Stop DML operations on the object.
2. Continue to compare the source object with the target object until they are both identical. This proves that Extract captured the remaining data changes from the transaction log and sent them to the target.
3. Perform the DDL.
4. Resume DML operations.

Comments in SQL

If a source DDL statement contains a comment in the middle of an object name, that comment will appear at the end of the object name in the target DDL statement. For example:

Source	Target
`CREATE TABLE hr./*comment*/emp ...`	`CREATE TABLE hr.emp /*comment*/ ...`

This does not affect the integrity of DDL synchronization. Comments in any other area of a DDL statement remain in place when replicated.

Compilation errors

If a CREATE operation on a trigger, procedure, function, or package results in compilation errors, Oracle GoldenGate executes the DDL operation on the target anyway. Technically, the DDL operations themselves completed successfully and should be propogated to allow dependencies to be executed on the target, for example in recursive procedures.

Interval partitioning

DDL replication is unaffected by interval partitioning, because the DDL is implicit.

Configuration guidelines for DDL support

Database privileges

See the Oracle GoldenGate *Oracle Installation and Setup Guide* for database privileges that are required for Oracle GoldenGate to support DDL capture and replication.

Initial synchronization

- To configure DDL replication, start with a target database that is synchronized with the source database. DDL support is compatible with the Replicat initial load method.
- 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.
- After initial synchronization of the source and target data, use all of the source sequence values at least once with NEXTVAL before you run the source applications. You can use a script that selects NEXTVAL from every sequence in the system. This must be done while Extract is running.

Process topology

- If using parallel Extract and/or Replicat processes, keep related DDL and DML together in the same process stream to ensure data integrity. Configure the processes so that:
    - all DDL and DML for any given object are processed by the same Extract group and by the same Replicat group.
    - all objects that are relational to one another are processed by the same process group.

  For example, if ReplicatA processes DML for Table1, then it should also process the DDL for Table1. If Table2 has a foreign key to Table1, then its DML and DDL operations also should be processed by ReplicatA.
- If an Extract group writes to multiple trails that are read by different Replicat groups, Extract sends all of the DDL to all of the trails. Use each Replicat group to filter the DDL by using the filter options of the DDL parameter in the Replicat parameter file.

Object names

- Oracle GoldenGate supports object names that contain multibyte characters and special alphanumeric characters, such as !, $, and #. Limitations apply when objects are mapped with TABLE or MAP parameters, because those parameters do not support all of the possible special characters. DDL on objects in MAP and TABLE statements inherit the limitations of those parameters. For more information about these parameters, see the documentation for MAP and TABLE in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- You can use standard Oracle GoldenGate asterisk wildcards (*) to specify object names in configuration parameters that support DDL synchronization. To process wildcards correctly, the WILDCARDRESOLVE parameter is set to DYNAMIC by default. If WILDCARDRESOLVE is set to anything else, the Oracle GoldenGate process that is processing DDL operations will abend and write the error to the process report.
- You can use the asterisk (*) wildcard to specify Oracle schema names in configuration parameters that support DDL synchronization. This feature is disabled by default. To enable it, use the _ALLOWWILDCARDSCHEMAS parameter in the GLOBALS parameter file. This is an unpublished parameter. Please contact Oracle Support before using this parameter.
- Oracle GoldenGate supports the use of a space before, after, or both before and after, the dot that separates owner and object names in a DDL statement. Only one space on each side of the dot is supported. For example, the following are valid:

```
CREATE TABLE fin . customers...
CREATE TABLE fin. customers...
CREATE TABLE fin .customers...
```

Data continuity after CREATE or RENAME

To replicate DML operations on new Oracle tables resulting from a CREATE or RENAME operation, the names of the new tables must be specified in TABLE and MAP statements in the parameter files. You can use wildcards to make certain that they are included.

To create a new user with CREATE USER and then move new or renamed tables into that schema, the new user name must be specified in TABLE and MAP statements. To create a new user "fin2" and move new or renamed tables into that schema, the parameter statements could look as follows, depending on whether you want the "fin2" objects mapped to the same, or different, schema on the target:

Extract:

```
TABLE fin2.*;
```

Replicat:

```
MAP fin2*, TARGET <different_schema>.*;
```

Understanding DDL scopes

Database objects are classified into *scopes*. A scope is a category that defines how DDL operations on an object are handled by Oracle GoldenGate. The scopes are:

- MAPPED
- UNMAPPED
- OTHER

The use of scopes enables granular control over the filtering of DDL operations, string substitutions, and error handling.

Mapped scope

Objects that are specified in TABLE and MAP statements are of *MAPPED scope*. Extraction and replication instructions in those statements apply to both data (DML) and DDL on the specified objects, unless override rules are applied.

For objects in TABLE and MAP statements, the DDL operations listed in the following table are supported.

Table 14 Objects that can be mapped in MAP and TABLE statements

Operations	Object[1]
CREATE	TABLE (includes AS SELECT)
ALTER	INDEX
DROP	TRIGGER
RENAME	SEQUENCE
COMMENT ON[2]	MATERIALIZED VIEW
	VIEW
	FUNCTION
	PACKAGE
	PROCEDURE
	SYNONYM
	PUBLIC SYNONYM[3]
GRANT	TABLE
REVOKE	SEQUENCE
	MATERIALIZED VIEW
ANALYZE	TABLE
	INDEX
	CLUSTER

[1] TABLE and MAP do not support some special characters that could be used in an object name affected by these operations. For a list of those characters, see the MAP and TABLE parameter descriptions in the Oracle GoldenGate *Windows and UNIX Reference Guide*. Objects with non-supported special characters are supported by the scopes of UN-MAPPED and OTHER.

[2] Applies to COMMENT ON TABLE, COMMENT ON COLUMN.

[3] Table name must be qualified with schema name.

For Extract, MAPPED scope marks an object for DDL capture according to the instructions in the TABLE statement. For Replicat, MAPPED scope marks DDL for replication and maps it to the object specified by the schema and name in the TARGET clause of the MAP statement. To perform this mapping, Replicat issues ALTER SESSION to set the schema of the Replicat session to the schema that is specified in the TARGET clause. If the DDL contains unqualified objects, the schema that is assigned on the target depends on circumstances described in "Correctly identifying unqualified object names in DDL" on page 150.

Assume the following TABLE and MAP statements:

Extract (source)	**Replicat (target)**
`TABLE fin.expen;`	`MAP fin.expen, TARGET fin2.expen2;`
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.bak_*;`

Also assume a source DDL statement of:

```
ALTER TABLE fin.expen ADD notes varchar2(100);
```

In this example, because the source table "fin.expen" is in a MAP statement with a TARGET clause that maps to a different owner and table name, the target DDL statement becomes:

```
ALTER TABLE fin2.expen2 ADD notes varchar2(100);
```

Likewise, the following source and target DDL statements are possible for the second set of TABLE and MAP statements in the example:

Source:	`CREATE TABLE hr.tabPayables ... ;`
Target:	`CREATE TABLE hrBackup.bak_tabPayables ...;`

When objects are of MAPPED scope, you can omit their names from the DDL configuration parameters, unless you want to refine their DDL support further. If you ever need to change the object names in TABLE and MAP statements, the changes will apply automatically to the DDL on those objects.

If you include an object in a TABLE statement, but not in a MAP statement, the DDL for that object is MAPPED in scope on the source but UNMAPPED in scope on the target.

Mapping Oracle cluster tables and UDTs

An Oracle clustered table or Oracle user defined type (UDT) cannot be mapped to a different target name, but it can be mapped to a different target owner. Because these special kinds of objects can consist of underlying tables that, themselves, could be a mix of both MAPPED and UNMAPPED scope, name mapping cannot be used.

Mapping ALTER INDEX

An ALTER INDEX...RENAME command cannot be mapped to a different target index name, but it can be mapped to a different target owner.

Valid example:

```
ALTER INDEX src.ind RENAME TO indnew;
```

This DDL can be mapped with wildcards as:

```
MAP src.* TARGET tgt.*;
```

Alternatively, it can be mapped explicitly as the following, making sure to use the original index name in the source and target specifications:

```
MAP src.ind TARGET tgt.ind;
```

In either of the preceding cases, the target DDL will be:

```
ALTER INDEX tgt.ind RENAME TO indnew;
```

Invalid example:

A MAP statement such as the following is not valid:

```
MAP src.ind TARGET tgt.indnew;
```

That statement maps the old name to the new name, and the target DDL will become:

```
ALTER INDEX tgt.indnew RENAME TO indnew;
```

Unmapped scope

If a DDL operation is supported for use in a TABLE or MAP statement, but its base object name is not included in one of those parameters, it is of *UNMAPPED scope*.

An object name can be of UNMAPPED scope on the source (not in an Extract TABLE statement), but of MAPPED scope on the target (in a Replicat MAP statement), or the other way around. When Oracle DDL is of UNMAPPED scope in the Replicat configuration, Replicat will by default do the following:

1. Set the current owner of the Replicat session to the owner of the source DDL object.
2. Execute the DDL as that owner.
3. Restore Replicat as the current owner of the Replicat session.

See also "Correctly identifying unqualified object names in DDL" on page 150.

Other scope

DDL operations that cannot be mapped are of *OTHER scope*. When DDL is of OTHER scope in the Replicat configuration, it is applied to the target with the same owner and object name as in the source DDL.

An example of OTHER scope is a DDL operation that makes a system-specific reference, such as DDL that operates on data file names.

Some other examples of OTHER scope:

```
CREATE USER joe IDENTIFIED by joe;
CREATE ROLE ggs_gguser_role IDENTIFIED GLOBALLY;
ALTER TABLESPACE gg_user TABLESPACE GROUP gg_grp_user;
```

See also "Correctly identifying unqualified object names in DDL" on page 150.

Correctly identifying unqualified object names in DDL

Oracle DDL can contain object names that are not qualified with schema names. For example, in the following DDL the TAB table in the CREATE TABLE clause is unqualified:

```
ALTER SESION SET CURRENT_SCHEMA = SRC;
CREATE TABLE tab (X NUMBER);
CREATE TABLE SRC1.tab (X NUMBER) AS SELECT * FROM tab;
```

By default, any unqualified object in an Oracle DDL statement assumes the session schema, which can be either of the following:

- The schema of the user that started the SQL session.
- The schema that is set with an ALTER SESSION SET CURRENT_SCHEMA command. In the preceding example, SRC becomes the owner of the unqualified TAB table.

To replicate DDL that contains unqualified objects, Replicat does the following:

- If the unqualified object is of MAPPED scope (that is, its name satisfies a MAP specification), Replicat does one of the following:
    - If the actual schema of the unqualified object is the same as the source session schema, Replicat sets the schema to the schema that is specified in the TARGET clause of the MAP statement.
    - If the actual schema of the unqualified object is different from the source session schema, Replicat sets the schema to the source session schema.
- If the unqualified object is of UNMAPPED or OTHER scope, Replicat sets the schema to that of the source session schema.

You can map a source session schema to a different target session schema. Session schema mapping might be required for some DDL to succeed on the target, such as CREATE TABLE AS SELECT. This mapping is global and will override any other mappings that involve the same schema names. To map session schemas, use the DDLOPTIONS parameter with the MAPSESSIONSCHEMA option. For more information, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

You can prevent explicit schema mapping with the NOEXPLICITSCHEMAMAPPING option of the DDLOPTIONS parameter. See the DDLOPTIONS parameter documentation in the *Windows and UNIX Reference Guide*.

Enabling DDL support

By default, the status of DDL replication support is as follows:

- On the source, Oracle GoldenGate DDL support is disabled by default. You must configure Extract to capture DDL by using the DDL parameter.
- On the target, DDL support is enabled by default, to maintain the integrity of transactional data that is replicated. By default, Replicat will process all DDL operations that the trail contains. If needed, you can use the DDL parameter to configure Replicat to ignore or filter DDL operations.

Filtering DDL replication

For Oracle databases, you can use the following methods to filter DDL operations so that specific (or all) DDL is applied to the target database according to your requirements.

- *Filter with the DDL trigger on the source system*. This method makes use of an Oracle function that is called at run-time by the trigger when a DDL operation occurs. Information about the DDL is passed to this function, and you can use it to compute whether the DDL will be passed to Extract or not. (By default, all DDL is passed to

Extract.) By sending fewer DDL operations to Extract, this method can improve capture performance and can also be used for any other purpose that requires filtering at an early stage of processing.

- *Filter with the* DDL *parameter on the source, the target, or both*. This method is performed within Oracle GoldenGate, and both Extract and Replicat can execute filter criteria. Extract can perform filtering, or it can send all of the DDL to a trail, and then Replicat can perform the filtering. Alternatively, you can filter in a combination of different locations. The DDL parameter gives you control over where the filtering is performed, and it also offers more filtering options than the trigger method, including the ability to filter collectively based on the DDL scope (for example, include all MAPPED scope).
- *Combine trigger and DDL parameter filtering*. Any DDL that is passed to Extract after it is filtered by the DDL trigger can be filtered further with the DDL parameter to meet specific needs.

Filtering at the trigger level

To filter DDL at the level of the DDL trigger, perform the following steps.

1. Copy the ddl_filter.sql file that is in the Oracle GoldenGate installation directory to a test machine where you can test the code that you will be writing.
2. Open the file for editing. It contains a PL/SQL function named filterDDL, which you can modify to specify filter criteria. The information that is passed to this function includes the owner of the DDL object, the name of the object, the object type, and the operation type. The user that executed the DDL is available in the ora_login_user variable. Write filter code for each type of DDL that you want to be included in, or excluded from, Extract processing.
3. (Optional) Write the code to set the variable getStatement to YES if you want to compute the leading 30K of DDL text in the stmt variable. By default DDL text is not computed, so as to prevent unnecessary overhead.
4. Write the code to compute the variable retVal to be either INCLUDE or EXCLUDE. This value determines whether or not the DDL operation is passed to Extract. The default is INCLUDE.
5. Save the code.
6. Stop DDL activity on the test system.
7. Compile the ddl_filter.sql file as follows:

   ```
   @ddl_filter schema_name
   ```

   Where: schema_name is the schema where the Oracle GoldenGate DDL objects are installed. See the Oracle GoldenGate *Oracle Installation and Setup Guide* for information on these objects.
8. Test in the test environment to make certain that the filtering works. It is important to perform this testing, because any errors in the code could cause source and target DDL to become out of synchronization.
9. After a successful test, copy the file to the Oracle GoldenGate installation directory on the source production system.
10. Stop DDL activity on the source system.

11. Compile the ddl_filter.sql file as you did before.

    ```
    @ddl_filter schema_name
    ```

12. Resume DDL activity on the source system.

Filtering with the DDL parameter

The DDL parameter is the main Oracle GoldenGate parameter for filtering DDL within the Extract and Replicat processes.

When used without options, the DDL parameter performs no filtering, and it causes all DDL operations to be propagated as follows:

- As an Extract parameter, it captures all supported DDL operations that are generated on all supported database objects and sends them to the trail.
- As a Replicat parameter, it replicates all DDL operations from the Oracle GoldenGate trail and applies them to the target. This is the same as the default behavior without this parameter.

When used with options, the DDL parameter acts as a filtering agent to include or exclude DDL operations based on:

- scope
- object type
- operation type
- object name
- strings in the DDL command syntax or comments, or both

Only one DDL parameter can be used in a parameter file, but you can combine multiple inclusion and exclusion options to filter the DDL to the required level.

- DDL filtering options are valid for a primary Extract that captures from the transaction source, but not for a data-pump Extract.
- When combined, multiple filter option specifications are linked logically as "AND" statements.
- All filter criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex DDL filtering criteria, it is recommended that you test your configuration in a test environment before using it in production.

WARNING Do not include any Oracle GoldenGate-installed DDL objects in a DDL parameter, in a TABLE parameter, or in a MAP parameter, nor in a TABLEEXCLUDE or MAPEXCLUDE parameter. Make certain that wildcard specifications in those parameters do not include Oracle GoldenGate-installed DDL objects. These objects must not be part of the Oracle GoldenGate configuration, but the Extract process must be aware of operations on them, and that is why you must not explicitly exclude them from the configuration with an EXCLUDE, TABLEEXCLUDE, or MAPEXCLUDE parameter statement.

NOTE Before you create a DDL parameter statement, it might help to review "How DDL is evaluated for processing" in this chapter.

Syntax

```
DDL [
{INCLUDE | EXCLUDE}
    [, MAPPED | UNMAPPED | OTHER | ALL]
    [, OPTYPE <type>]
    [, OBJTYPE '<type>']
    [, OBJNAME "<name>"]
    [, INSTR '<string>']
    [, INSTRCOMMENTS '<comment_string>']
]
[...]
```

Table 15 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME "hr.*"`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME "hr.*"`
	`DDL INCLUDE OBJNAME "fin.*" EXCLUDE "fin.ss"`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.

Table 15 DDL inclusion and exclusion options

Option	Description
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	In the following example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.

Table 15 DDL inclusion and exclusion options

Option	Description

	For DDL that creates triggers, synonyms, and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger, synonym, or index.
	For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.
	INSTRCOMMENTS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

Table 15 DDL inclusion and exclusion options

Option	Description

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. INSTRCOMMENTSWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Combining DDL parameter options

The following is an example of how to combine DDL parameter options.

```
DDL  &
INCLUDE UNMAPPED &
   OPTYPE alter &
   OBJTYPE 'table' &
   OBJNAME "users.tab*" &
INCLUDE MAPPED OBJNAME "*" &
EXCLUDE MAPPED OBJNAME "temporary.tab*"
```

The combined filter criteria in this statement specify the following:

- INCLUDE all ALTER TABLE statements for tables that are not mapped with a TABLE or MAP statement (UNMAPPED scope),
  - only if those tables are owned by "users" and their names start with "tab,"
- and INCLUDE all DDL operation types for all tables that are mapped with a TABLE or MAP statement (MAPPED scope).

- and EXCLUDE all DDL operation types for all tables that are MAPPED in scope,
  - only if those tables are owned by "temporary."
  - and only if their names begin with "tab."

Special filter cases

The following are special cases that you should be aware of when creating your filter conditions.

DDL EXCLUDE ALL

DDL EXCLUDE ALL is a special processing option that maintains up-to-date object metadata for Oracle GoldenGate, while blocking the replication of the DDL operations themselves. You can use DDL EXCLUDE ALL when using a method other than Oracle GoldenGate to apply DDL to the target, but you want Oracle GoldenGate to replicate data changes to the target objects. It provides the current metadata to Oracle GoldenGate as objects change, thus preventing the need to stop and start the Oracle GoldenGate processes. The following special conditions apply to DDL EXCLUDE ALL:

- DDL EXCLUDE ALL does not require the use of an INCLUDE clause.
- When using DDL EXCLUDE ALL, you may set the WILDCARDRESOLVE parameter to IMMEDIATE to allow immediate DML resolution if required.

To prevent all DDL metadata and operations from being replicated, omit the DDL parameter entirely. The DDL trigger will continue to record the DDL operations to the history table, unless disabled manually.

Implicit DDL

User-generated DDL operations can generate implicit DDL operations. For example, the following statement causes the Oracle DDL trigger to process two distinct DDL operations.

```
CREATE TABLE customers (custID number, name varchar2(50), address
varchar2(75), address2 varchar2(75), city varchar2(50), state (varchar2(2),
zip number, contact varchar2(50), areacode number(3), phone number(7), primary
key (custID));
```

- The first (explicit) DDL operation is the CREATE TABLE statement itself.
- The second DDL operation is an implicit CREATE UNIQUE INDEX statement that creates the index for the primary key. This operation is generated by the database engine, not a user application.

Guidelines for filtering implicit DDL

When the *DDL parameter* is used to filter DDL operations, Oracle GoldenGate filters out any implicit DDL by default, because the explicit DDL will generate the implicit DDL on the target. For example, the target database will create the appropriate index when the CREATE TABLE statement in the preceding example is applied by Replicat.

However, when the *DDL trigger* is used to filter DDL operations, you must handle the implicit DDL in your filter rules based on the following:

- If your filtering rules exclude the explicit DDL from being propagated, you must also create a rule to exclude the implicit DDL. For example, if you exclude the CREATE TABLE statement in the preceding example, but do not exclude the CREATE UNIQUE INDEX statement, the target database will try to create the index on a non-existent table.
- If your filtering rules permit the propagation of the explicit DDL, you do not need to exclude the implicit DDL. It will be handled correctly by Oracle GoldenGate and the target database.

How Oracle GoldenGate handles derived object names

DDL operations can contain a *base object* name and also a *derived object* name. A base object is an object that contains data. A derived object is an object that inherits some attributes of the base object to perform a function related to that object. DDL statements that have both base and derived objects are:

- RENAME and ALTER RENAME
- CREATE and DROP on an index, synonym, or trigger

Consider the following DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

In this case, the table is the base object. Its name (hr.tabPayroll) is the *base name* and is subject to mapping with TABLE or MAP under the MAPPED scope. The derived object is the index, and its name (hr.indexPayrollDate) is the *derived name*.

You can map a derived name in its own TABLE or MAP statement, separately from that of the base object. Or, you can use one MAP statement to handle both. In the case of MAP, the conversion of derived object names on the target works as follows.

MAP exists for base object, but not derived object

If there is a MAP statement for the base object, but not for the derived object, the result is an *implicit mapping* of the derived object. Assuming the DDL statement includes MAPPED, Replicat gives the derived object the same target owner as that of the base object. The name of the derived object stays the same as in the source statement. For example, assume the following:

Extract (source)	Replicat (target)
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.*;`

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The CREATE INDEX statement is executed by Replicat on the target as:

```
CREATE INDEX hrBackup.indexPayrollDate ON TABLE hrBackup.tabPayroll
(payDate);
```

The rule for the implicit mapping is based the typical industry practice of giving derived objects the same owner as the base object. It ensures the correct name conversion even if the name of the derived object is not fully qualified in the source statement. Also, when indexes are owned by the same target owner as the base object, an implicit mapping eliminates the need to map derived object names explicitly.

MAP exists for base and derived objects

If there is a MAP statement for the base object and also one for the derived object, the result is an *explicit mapping*. Assuming the DDL statement includes MAPPED, Replicat converts the owner and name of each object according to its own TARGET clause. For example, assume the following:

Extract (source)	**Replicat (target)**
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.*;`
`TABLE hr.index*;`	`MAP hr.index*, TARGET hrIndex.*;`

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The CREATE INDEX statement is executed by Replicat on the target as:

```
CREATE INDEX hrIndex.indexPayrollDate ON TABLE hrBackup.tabPayroll
(payDate);
```

Use an explicit mapping when the index on the target must be owned by a different owner from that of the base object, or when the name on the target must be different from that of the source.

MAP exists for derived object, but not base object

If there is a MAP statement for the derived object, but not for the base object, Replicat does not perform any name conversion for either object. The target DDL statement is the same as that of the source. To map a derived object, the choices are:

- Use an explicit MAP statement for the base object.
- If names permit, map both base and derived objects in the same MAP statement by means of a wildcard.
- Create a MAP statement for each object, depending on how you want the names converted.

New tables as derived objects

The following explains how Oracle GoldenGate handles new tables that are created from:

- RENAME and ALTER RENAME
- CREATE TABLE AS SELECT

RENAME and ALTER TABLE RENAME

In RENAME and ALTER TABLE RENAME operations, the base object is always the new table name. In the following example, the base object name is considered to be "index_paydate."

```
ALTER TABLE hr.indexPayrollDate RENAME TO index_paydate;
```

or...

```
RENAME hr.indexPayrollDate TO index_paydate;
```

The derived object name is "hr.indexPayrollDate."

See "Controlling whether renames enter the DDL configuration" on page 170 for additional information on renames as they relate to DDL replication.

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT statements include SELECT statements and INSERT statements that affect any number of underlying objects. On the target, Oracle GoldenGate obtains the data for the AS SELECT clause from the target database. The objects in the AS SELECT clause must exist in the target database, and their names must be identical to the ones on the source.

In a MAP statement, Oracle GoldenGate only maps the name of the new table (CREATE TABLE <name>) to the TARGET specification, but does not map the names of the underlying objects from the AS SELECT clause. There could be dependencies on those objects that could cause data inconsistencies if the names were converted to the TARGET specification.

The following shows an example of a CREATE TABLE AS SELECT statement on the source and how it would be replicated to the target by Oracle GoldenGate.

```
CREATE TABLE a.tab1 AS SELECT * FROM a.tab2;
```

The MAP statement for Replicat is:

```
MAP a.tab*, TARGET a.x*;
```

The target DDL statement that is applied by Replicat is this:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.tab2;
```

not this:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.xtab2;
```

The name of the table in the AS SELECT * FROM clause remains as it was on the source: tab2.

To keep the data in the underlying objects consistent on source and target, you can configure them for data replication by Oracle GoldenGate. In the preceding example, you could use the following statements to accommodate this requirement:

Source	Target

`TABLE a.tab*;`	`MAPEXCLUDE a.tab2`
	`MAP a.tab*, TARGET a.x*;`
	`MAP a.tab2, TARGET a.tab2;`

See also "Correctly identifying unqualified object names in DDL" on page 150.

Disabling the mapping of derived objects

Use the DDLOPTIONS parameter with the NOMAPDERIVED option to prevent the conversion of the name of a derived object according to a TARGET clause of a MAP statement that includes it. NOMAPDERIVED overrides any explicit MAP statements that contain the name of the base or derived object. Source DDL that contains derived objects is replicated to the target with the same owner and object names as on the source.

The following table shows the results of MAPDERIVED compared to NOMAPDERIVED, based on whether there is a MAP statement just for the base object, just for the derived object, or for both.

Table 16 [NO]MAPDERIVED results on target based on mapping configuration

Base Object	Derived Object	MAP/NOMAP DERIVED?	Derived object converted per a MAP?	Derived object gets owner of base object?
mapped[1]	mapped	MAPDERIVED	yes	no
mapped	not mapped	MAPDERIVED	no	yes
not mapped	mapped	MAPDERIVED	no	no
not mapped	not mapped	MAPDERIVED	no	no
mapped	mapped	NOMAPDERIVED	no	no
mapped	not mapped	NOMAPDERIVED	no	no
not mapped	mapped	NOMAPDERIVED	no	no
not mapped	not mapped	NOMAPDERIVED	no	no

[1] Mapped means included in a MAP statement.

The following examples illustrate the results of MAPDERIVED as compared to NOMAPDERIVED.

In the following table, both trigger and table are owned by "rpt" on the target because both base and derived names are converted by means of MAPDERIVED.

Table 17 Default mapping of derived object names (MAPDERIVED)

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER rpt.act_trig ON rpt.acct;`

In the following table, the trigger is owned by "fin," because conversion is prevented by means of NOMAPDERIVED.

Table 18 Mapping of derived object names when using NOMAPDERIVED

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER fin.act_trig ON rpt.acct;`

NOTE In the case of a RENAME statement, the new table name is considered to be the base table name, and the old table name is considered to be the derived table name.

Using DDL string substitution

You can substitute strings within a DDL operation while it is being processed by Oracle GoldenGate. This feature provides a convenience for changing and mapping directory names, comments, and other things that are not directly related to data structures. For example, you could substitute one tablespace name for another, or substitute a string within comments. String substitution is controlled by the DDLSUBST parameter.

Guidelines for using DDLSUBST

- Do not use DDLSUBST to convert column names and data types to something different on the target. Changing the structure of a target object in this manner will cause errors when data is replicated to it. Likewise, do not use DDLSUBST to change owner and table names in a target DDL statement. Always use a MAP statement to map a replicated DDL operation to a different target object.
- DDLSUBST always executes after the DDL parameter, regardless of their relative order in the parameter file. Because the filtering executes first, use filtering criteria that is compatible with the criteria that you are using for string substitution. For example, consider the following parameter statements:

  ```
  DDL INCLUDE OBJNAME "fin.*"
  DDLSUBST 'cust' WITH 'customers' INCLUDE OBJNAME "sales.*"
  ```

  In this example, no substitution occurs because the objects in the INCLUDE and DDLSUBST statements are different. The fin-owned objects are included in the Oracle GoldenGate DDL configuration, but the sales-owned objects are not.
- You can use multiple DDLSUBST parameters. They execute in the order listed in the parameter file.
- For Oracle DDL that includes comments, do not use the DDLOPTIONS parameter with the REMOVECOMMENTS BEFORE option if you will be doing string substitution on those comments. REMOVECOMMENTS BEFORE removes comments before string substitution occurs. To remove comments, but allow string substitution, use the REMOVECOMMENTS AFTER option.
- There is no maximum string size for substitutions, other than the limit that is imposed by the database. If the string size exceeds the database limit, the Extract or Replicat process that is executing the operation abends.

NOTE Before you create a DDLSUBST parameter statement, it might help to review "How DDL is evaluated for processing" in this chapter.

Syntax

```
DDLSUBST '<search_string>' WITH '<replace_string>'
[INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>]
```

Argument	Description
`'<search_string>'`	The string in the source DDL statement that you want to replace. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.
`WITH`	Required keyword.
`'<replace_string>'`	The string that you want to use as the replacement in the target DDL. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.
`INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>`	Use one or more INCLUDE and EXCLUDE statements to filter the DDL operations for which the string substitution rules are applied. See the following table.

Table 19 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME "hr.*"`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME "hr.*"`
	`DDL INCLUDE OBJNAME "fin.*" EXCLUDE "fin.ss"`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

Table 19 DDL inclusion and exclusion options

Option	Description
`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`

Table 19 DDL inclusion and exclusion options

Option	Description
	In the following example, a CREATE TABLE statement executes like this on the source: `CREATE TABLE fin.exp_phone;` And like this on the target: `CREATE TABLE fin2.exp_phone;` If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter. For DDL that creates triggers, synonyms, and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger, synonym, or index. For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig." `CREATE TRIGGER hr.insert_trig ON hr.accounts;` For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct." `ALTER TABLE hr.accounts RENAME TO acct;`
`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.
	INSTRCOMMENTS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

Table 19 DDL inclusion and exclusion options

Option	Description

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. INSTRCOMMENTSWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Example In the following example, a new directory is substituted only if the DDL command includes the word "logfile." If the search string is found multiple times, the replacement string is inserted multiple times.

```
DDLSUBST '/file1/location1' WITH '/file2/location2'
INCLUDE INSTR'logfile'
```

Example In the following example, the string 'cust' is replaced with the string 'customers' for tables owned by "fin".

```
DDLSUBST 'cust' WITH 'customers'
INCLUDE ALL OBJTYPE 'table' OBJNAME "fin.*"
```

The search is not case-sensitive. To represent a quotation mark in a string, use a double quotation mark.

Example This example uses multiple DDLSUBST parameters. They will execute in the order listed in the parameter file. The net effect is to substitute “a” and “b” strings with “c.”

```
DDLSUBST ‘a’ WITH ‘b’ INCLUDE ALL
DDLSUBST ‘b’ WITH ‘c’ INCLUDE ALL
```

Controlling the propagation of DDL that is executed by Replicat

Extract and Replicat both issue DDL operations.

- Extract issues ALTER TABLE statements to create log groups,
- Replicat applies replicated DDL statements to the target.

To identify Oracle GoldenGate DDL operations, the following comment is part of each Extract and Replicat DDL statement:

```
/* GOLDENGATE_DDL_REPLICATION */
```

The DDLOPTIONS parameter controls whether or not Replicat’s DDL is propagated.

- The GETREPLICATES and IGNOREREPLICATES options control whether *Replicat’s* DDL operations are captured by Extract or ignored. The default is IGNOREREPLICATES.
- The GETAPPLOPS and IGNOREAPPLOPS options control whether DDL from *applications other than Replicat* (the business applications) are captured or ignored.

By default, Extract ignores DDL that is applied to the local database by a local Replicat, so that the DDL is not sent back to its source, but Extract captures all other DDL that is configured for replication. The following is the default DDLOPTIONS configuration.

```
DDLOPTIONS GETAPPLOPS, IGNOREREPLICATES
```

This behavior can be modified. See the following topics:

- “Propagating DDL in an active-active (bi-directional) configurations”
- “Propagating DDL in a cascading configuration”

Propagating DDL in an active-active (bi-directional) configurations

Oracle GoldenGate supports active-active DDL replication between two systems. For an active-active bi-directional replication, the following must be configured in the Oracle GoldenGate processes:

1. DDL that is performed by a business application on one system must be replicated to the other system to maintain synchronization. To satisfy this requirement, include the GETAPPLOPS option in the DDLOPTIONS statement in the Extract parameter files on both systems.
2. DDL that is applied by Replicat on one system must be captured by the local Extract and sent back to the other system. To satisfy this requirement, use the GETREPLICATES option in the DDLOPTIONS statement in the Extract parameter files on both systems.

   NOTE An internal Oracle GoldenGate token will cause the actual Replicat DDL statement itself to be ignored to prevent loopback. The purpose of propagating Replicat DDL back to the original system is so that the Replicat on that system can update its object metadata cache, in preparation to receive incoming DML, which will have the new metadata. See the illustration in this topic.

3. Each Replicat must be configured to update its object metadata cache whenever the remote Extract sends over a captured Replicat DDL statement. To satisfy this requirement, use the UPDATEMETADATA option in the DDLOPTIONS statement in the Replicat parameter files on both systems.

The resultant DDLOPTIONS statements should look as follows:

Extract (primary and secondary)

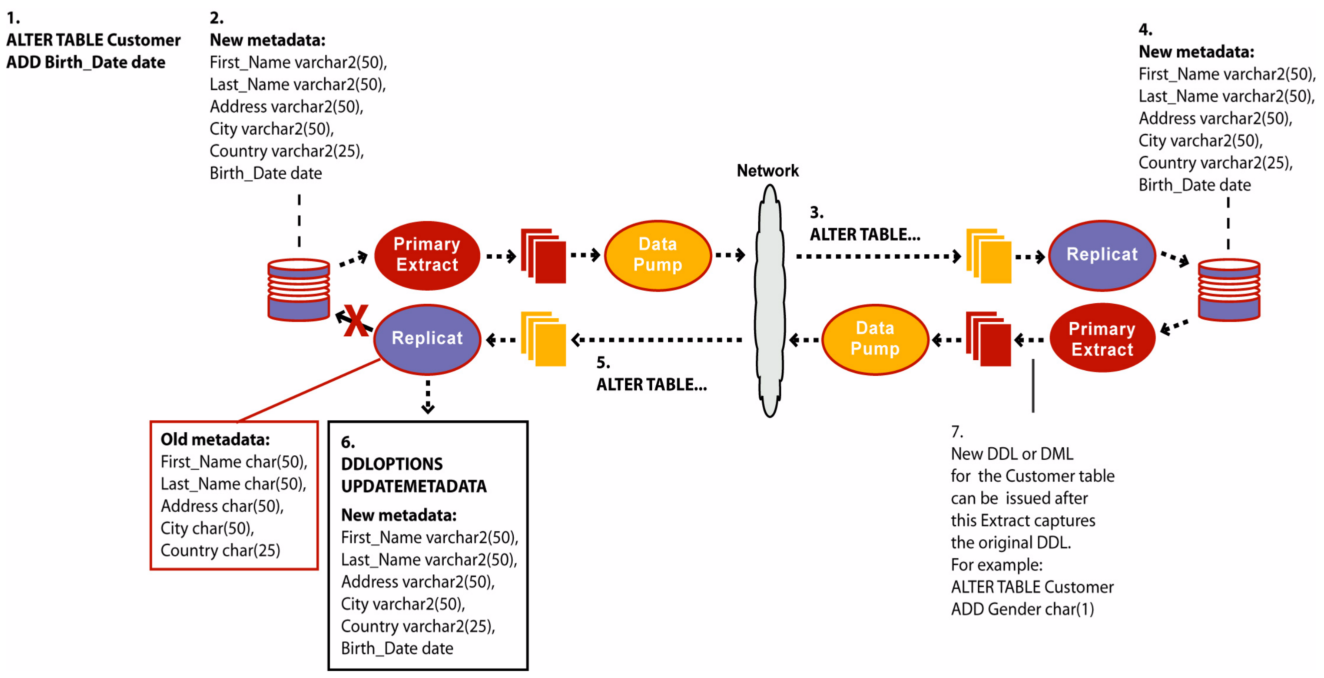
```
DDLOPTIONS GETREREPLICATES, GETAPPLOPS
```

Replicat (primary and secondary)

```
DDLOPTIONS UPDATEMETADATA
```

WARNING Before you allow new DDL or DML to be issued for the same object(s) as the original DDL, allow time for the original DDL to be replicated to the remote system and then captured again by the Extract on that system. This will ensure that the operations arrive in correct order to the Replicat on the original system, to prevent DML errors caused by metadata inconsistencies. See the diagram for more information.

Figure 18 Path of DDL in round trip to update Replicat object metadata cache



Propagating DDL in a cascading configuration

In a cascading configuration, use the following setting for DDLOPTIONS in the Extract parameter file on each intermediary system. This configuration forces Extract to capture the DDL from Replicat on an intermediary system and cascade it to the next system downstream.

```
DDLOPTIONS GETREPLICATES, IGNOREAPPLOPS
```

Adding supplemental log groups automatically

You can use the DDLOPTIONS parameter with the ADDTRANDATA option to:

- enable Oracle's supplemental logging automatically for new tables created with a CREATE TABLE.
- update Oracle's supplemental logging for tables affected by an ALTER TABLE to add or drop columns.
- update Oracle's supplemental logging for tables that are renamed.
- update Oracle's supplemental logging for tables where unique or primary keys are added or dropped.

By default, the ALTER TABLE that adds the supplemental logging is not replicated to the target unless the GETREPLICATES parameter is in use.

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

Removing comments from replicated DDL

You can use the DDLOPTIONS parameter with the REMOVECOMMENTS BEFORE and REMOVECOMMENTS AFTER options to prevent comments that were used in the source DDL from being included in the target DDL. By default, comments are not removed, so that they can be used for string substitution.

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

Controlling whether renames enter the DDL configuration

You can use the DDLOPTIONS parameter with the NOCROSSRENAME option to enforce the rule that objects which are excluded from the Oracle GoldenGate configuration cannot be renamed to names that are in the configuration. This is an example of how a rename could occur:

- TableA is excluded, but tableB is included.
- TableA gets renamed to tableB.

If an object does get renamed to one that is in the Oracle GoldenGate configuration, Extract issues a warning, so that you can take the appropriate action (keep it in the Oracle GoldenGate configuration or make the appropriate parameter adjustments to exclude it). An example of how this notification is useful is to prevent errors if a renamed object has a structure that is not supported by Oracle GoldenGate.

In an Oracle RAC environment, NOCROSSRENAME has the additional benefit of improving performance. It eliminates the processing overhead that otherwise is required across the nodes to keep track of excluded objects in case they are renamed to a name that is included.

NOCROSSRENAME applies globally to:

- all objects specified in TABLE and TABLEEXCLUDE statements in the parameter file

- all objects that are excluded from the Oracle GoldenGate DDL configuration, and not specified by TABLE or TABLEEXCLUDE.

DDLOPTIONS NOCROSSRENAME provides the same results as the TABLEEXCLUDE parameter, when used with the NORENAME option. The difference between the two parameters is that TABLEEXCLUDE NORENAME allows more selectivity than NOCROSSRENAME, because the functionality only affects the objects in the TABLEEXCLUDE statement.

Replicating an IDENTIFIED BY password

You can use the DDLOPTIONS parameter with the following options to control how the password of a replicated {CREATE | ALTER} USER <name> IDENTIFIED BY <password> statement is handled. These options must be used together.

DEFAULTUSERPASSWORD

This option is valid for Replicat. Use DEFAULTUSERPASSWORD for a replicated {CREATE | ALTER} USER <name> IDENTIFIED BY <password> statement to specify a different password from the one used in the source statement. By default, the source password is replicated to the target. You can provide a clear-text password or an encrypted password. Replicat will replace the placeholder that Extract writes to the trail with the specified password.

REPLICATEPASSWORD | NOREPLICATEPASSWORD

This option is valid for Extract. By default (REPLICATEPASSWORD), Oracle GoldenGate uses the source password from a {CREATE | ALTER} USER <name> IDENTIFIED BY <password> in the target CREATE or ALTER statement. To prevent the source password from being sent to the target, use NOREPLICATEPASSWORD.

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

How DDL is evaluated for processing

The following explains how Oracle GoldenGate processes DDL statements on the source and target systems. It shows the order in which different criteria in the Oracle GoldenGate parameters are processed, and it explains the differences between how Extract and Replicat each process the DDL.

Extract

1. Extract captures a DDL operation.
2. Extract separates comments, if any, from the main statement.
3. Extract searches for the DDL parameter. (This example assumes it exists.)
4. Extract searches for the IGNOREREPLICATES parameter. If it is present, and if Replicat produced this DDL operation on this system, Extract ignores the DDL statement. (This example assumes no Replicat operations on this system.)
5. Extract determines whether the statement is a RENAME. If so, the rename is flagged internally.
6. Extract gets the base object name and, if present, the derived object name.

7. If the statement is a RENAME, Extract changes it to ALTER TABLE RENAME.
8. Extract searches for the DDLOPTIONS REMOVECOMMENTS BEFORE parameter. If it is present, Extract removes the comments from the DDL statement, but stores them in case there is a DDL INCLUDE or DDL EXCLUDE clause that uses INSTR or INSTRCOMMENTS.
9. Extract determines the DDL scope: MAPPED, UNMAPPED or OTHER. It is MAPPED if:
   - the operation and object types are supported for mapping.

     and...
   - the base object name and/or derived object name (if RENAME) is in a TABLE parameter.

   It is UNMAPPED if:
   - the operation and object types are not supported for mapping.

     and...
   - the base object name and/or derived object name (if RENAME) is not in a TABLE parameter.

   Otherwise the operation is identified as OTHER.
10. Extract checks the DDL parameter for INCLUDE and EXCLUDE clauses, and it evaluates the DDL parameter criteria in those clauses. All options must evaluate to TRUE in order for the INCLUDE or EXCLUDE to evaluate to TRUE. The following occurs:
    - If an EXCLUDE clause evaluates to TRUE, Extract discards the DDL operation and evaluates another DDL operation. In this case, the processing steps start over.
    - If an INCLUDE clause evaluates to TRUE, or if the DDL parameter does not have any INCLUDE or EXCLUDE clauses, Extract includes the DDL operation, and the processing logic continues.
11. Extract searches for a DDLSUBST parameter and evaluates the INCLUDE and EXCLUDE clauses. If the criteria in those clauses add up to TRUE, Extract performs string substitution. Extract evaluates the DDL operation against each DDLSUBST statement in the parameter file. For all true DDLSUBST statements, Extract performs string substitution in the order that the DDLSUBST parameters are listed in the file.
12. Now that DDLSUBT has been processed, Extract searches for the REMOVECOMMENTS AFTER parameter. If it is present, Extract removes the comments from the DDL statement.
13. Extract searches for DDLOPTIONS ADDTRANDATA. If it is present, and if the operation is CREATE TABLE, Extract issues the ALTER TABLE <name> ADD SUPPLEMENTAL LOG GROUP command on the table.
14. Extract writes the DDL statement to the trail.

Replicat

1. Replicat reads the DDL operation from the trail.
2. Replicat separates comments, if any, from the main statement.
3. Replicat searches for DDLOPTIONS REMOVECOMMENTS BEFORE. If it is present, Replicat removes the comments from the DDL statement.
4. Replicat evaluates the DDL synchronization scope to determine if the DDL qualifies for name mapping. Anything else is of OTHER scope.

5. Replicat evaluates the MAP statements in the parameter file. If the source base object name for this DDL (as read from the trail) appears in any of the MAP statements, the operation is marked as MAPPED in scope. Otherwise it is marked as UNMAPPED in scope.
6. Replicat replaces the source base object name with the base object name that is specified in the TARGET clause of the MAP statement.
7. If there is a derived object, Replicat searches for DDLOPTIONS MAPDERIVED. If it is present, Replicat replaces the source derived name with the target derived name from the MAP statement.
8. Replicat checks the DDL parameter for INCLUDE and EXCLUDE clauses, and it evaluates the DDL parameter criteria contained in them. All options must evaluate to TRUE in order for the INCLUDE or EXCLUDE to evaluate to TRUE. The following occurs:
   - If any EXCLUDE clause evaluates to TRUE, Replicat discards the DDL operation and starts evaluating another DDL operation. In this case, the processing steps start over.
   - If any INCLUDE clause evaluates to TRUE, or if the DDL parameter does not have any INCLUDE or EXCLUDE clauses, Replicat includes the DDL operation, and the processing logic continues.
9. Replicat searches for the DDLSUBST parameter and evaluates the INCLUDE and EXCLUDE clauses. If the options in those clauses add up to TRUE, Replicat performs string substitution. Replicat evaluates the DDL operation against each DDLSUBST statement in the parameter file. For all true DDLSUBST statements, Replicat performs string substitution in the order that the DDLSUBST parameters are listed in the file.
10. Now that DDLSUBT has been processed, Replicat searches for the REMOVECOMMENTS AFTER parameter. If it is present, Replicat removes the comments from the DDL statement.
11. Replicat executes the DDL operation on the target database.
12. If there are no errors, Replicat processes the next DDL statement. If there are errors, Replicat performs the following steps.
13. Replicat analyzes the INCLUDE and EXCLUDE rules in the Replicat DDLERROR parameter statements in the order that they appear in the parameter file. If Replicat finds a rule for the error code, it applies the specified error handling; otherwise, it applies DEFAULT handling.
14. If the error handling does not enable the DDL operation to succeed, Replicat does one of the following: abends, ignores the operation, or discards it as specified in the rules.

NOTE If there are multiple targets for the same source in a MAP statement, the processing logic executes for each one.

Handling Extract DDL processing errors

Use the Extract option of the DDLERROR parameter to handle errors on objects found by Extract for which metadata cannot be found.

Syntax

```
DDLERROR [RESTARTSKIP <num skips>] [SKIPTRIGGERERROR <num errors>]
```

Where:

- RESTARTSKIP skips a number of DDL operations on startup to prevent Extract from abending on an error. By default, Extract abends on an error so that no operations are skipped. You can skip up to 100,000 DDL operations.

To write information about skipped operations to the Extract report file, use the DDLOPTIONS parameter with the REPORT option.

Handling Replicat DDL processing errors

Use the Replicat options of the DDLERROR parameter to handle errors that occur when DDL is applied to the target database. With DDLERROR options, you can handle most errors in a default manner, for example to stop processing, and also handle other errors in a specific manner. You can use multiple instances of DDLERROR in the same parameter file to handle all errors that are anticipated.

Use the combination of <error>, DEFAULT, and <response> to create rules for how Replicat responds to anticipated and unanticipated DDL errors. Make certain to specify the appropriate inclusion and exclusion clauses to apply the rules to the intended DDL. Then, use additional options to refine the error handling, as needed.

Syntax

```
DDLERROR
{<error> | DEFAULT} {<response>}
{INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>}
[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]
```

Argument	Description

`{<error> | DEFAULT} {<response>}`	◆ <error> is a specific DDL error that you want to be handled with this statement.
	◆ DEFAULT sets a global response to all DDL errors except those for which explicit DDLERROR statements are specified.
	◆ <response> can be one of the following:
	
	`ABEND`
	Rolls back the operation and terminates processing abnormally. ABEND is the default.
	
	`DISCARD`
	Logs the offending operation to the discard file but continue processing subsequent DDL. Specify a discard file with the DISCARDFILE parameter.

Argument	Description

	`IGNORE`
	Ignores the error.
	
	`RETRYOP MAXRETRIES <n> [RETRYDELAY <delay>]`
	Retries the offending operation. Use the MAXRETRIES option to control the number of retries. Replicat abends after the specified number of MAXRETRIES. Specify a whole integer.
	Use RETRYDELAY to set the amount of time, in seconds, between retry attempts.

`{INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>}`	Controls whether specific DDL is handled or not handled by the DDLERROR statement. See the following table for descriptions.
`[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]`	Controls whether or not Extract abends when DML is issued on objects that could not be found on the target. This condition is typically caused by DDL that is issued directly on the target outside of replication, or by a discrepancy between source and target definitions. IGNOREMISSINGOBJECTS causes Replicat to skip DML operations on missing tables. ABENDONMISSINGOBJECTS causes Replicat to abend on DML operations on missing tables.

Table 20 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME “hr.*”`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME “hr.*”`
	`DDL INCLUDE OBJNAME “fin.*” EXCLUDE “fin.ss”`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

Table 20 DDL inclusion and exclusion options

Option	Description
`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`

Table 20 DDL inclusion and exclusion options

Option	Description

	In the following example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers, synonyms, and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger, synonym, or index.
	For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (' ') that are within the string, nor does it support NULL values.

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.
	INSTRCOMMENTS does not support single quotation marks (' ') that are within the string, nor does it support NULL values.

Table 20 DDL inclusion and exclusion options

Option	Description

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. INSTRCOMMENTSWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Sample DDLERROR statement

In the following example, the DDLERROR statement causes Replicat to ignore the specified error, but not before trying the operation again three times at ten-second intervals. Replicat applies the error handling to DDL operations executed on objects whose names satisfy the wildcard of "tab*" (any user, any operation) except those that satisfy "tab1*."

```
DDLERROR <error> IGNORE RETRYOP MAXRETRIES 3 RETRYDELAY 10 &
INCLUDE ALL OBJTYPE TABLE OBJNAME "tab*" EXCLUDE OBJNAME "tab1*"
```

To handle all errors except that error, the following DDLERROR statement can be added.

```
DDLERROR DEFAULT ABENDS
```

In this case, Replicat abends on DDL errors.

Using multiple DDLERROR statements

The order in which you list DDLERROR statements in the parameter file does not affect their validity unless multiple DDLERROR statements specify the same error, without any additional qualifiers. In that case, Replicat only uses the first one listed. For example, given the following statements, Replicat will abend on the error.

```
DDLERROR <error1> ABEND
DDLERROR <error1> IGNORE
```

With the proper qualifiers, however, the previous configuration becomes a more useful one. For example:

```
DDLERROR <error1> ABEND INCLUDE OBJNAME "tab*"
DDLERROR <error1> IGNORE
```

In this case, because there is an INCLUDE statement, Replicat will abend only if an object name in an errant DDL statement matches wildcard "tab*." Replicat will ignore errant operations that include any other object name.

Handling DDL trigger errors

Use the following parameters in the params.sql non-executable script to handle failures of the Oracle GoldenGate DDL trigger in relation to whether the source DDL fails or succeeds.

- **ddl_fire_error_in_trigger**: If set to TRUE, failures of the Oracle GoldenGate DDL trigger are raised with a Oracle GoldenGate error message and a database error message to the source end-user application. The source operations fails.

  If set to FALSE, no errors are raised, and a message is written to the trigger trace file in the Oracle GoldenGate directory. The source operation succeeds, but no DDL is replicated. The target application will eventually fail if subsequent data changes do not match the old target object structure. The default is FALSE.
- **_ddl_cause_error**: If set to TRUE, tests the error response of the trigger by deliberately causing an error. To generate the error, Oracle GoldenGate attempts to SELECT zero rows without exception handling. Revert this flag to the default of FALSE after testing is done.

The params.sql script is in the root Oracle GoldenGate directory.

Viewing DDL report information

By default, Oracle GoldenGate shows basic statistics about DDL operations at the end of the Extract and Replicat reports. To enable expanded DDL reporting, use the DDLOPTIONS parameter with the REPORT option. Expanded reporting includes the following information about DDL processing:

- A step-by-step history of the DDL operations that were processed by Oracle GoldenGate.
- The DDL filtering and processing parameters that are being used.

Expanded DDL report information increases the size of the report file, but it might be useful in certain situations, such as for troubleshooting or to determine when an ADDTRANDATA to add supplemental logging was applied.

To view a process report

To view a report, use the VIEW REPORT command in GGSCI.

```
VIEW REPORT <group>
```

Extract DDL reporting

The Extract report lists the following:

- The entire syntax of each captured DDL operation, the start and end SCN, the Oracle instance, the DDL sequence number (from the SEQNO column of the history table), and the size of the operation in bytes.
- A subsequent entry that shows how processing criteria was applied to the operation, for example string substitution or INCLUDE and EXCLUDE filtering.
- Another entry showing whether the operation was written to the trail or excluded.

The following, taken from an Extract report, shows an included operation and an excluded operation. There is a report message for the included operation, but not for the excluded one.

```
2011-01-20 15:11:41  GGS INFO     2100  DDL found, operation [create table
myTable (
    myId number (10) not null,
    myNumber number,
    myString varchar2(100),
    myDate date,
    primary key (myId)
) ], start SCN [1186754], commit SCN [1186772] instance [test10g (1)], DDL
seqno [4134].

2011-01-20 15:11:41  GGS INFO    2100  DDL operation included [INCLUDE OBJNAME
myTable*], optype [CREATE], objtype [TABLE], objname [QATEST1.MYTABLE].

2011-01-20 15:11:41  GGS INFO    2100  DDL operation written to extract trail
file.

2011-01-20 15:11:42  GGS INFO    2100  Successfully added TRAN DATA for table
with the key, table [QATEST1.MYTABLE], operation [ALTER TABLE
"QATEST1"."MYTABLE" ADD SUPPLEMENTAL LOG GROUP "GGS_MYTABLE_53475" (MYID)
ALWAYS  /* GOLDENGATE_DDL_REPLICATION */ ].

2011-01-20 15:11:43  GGS INFO     2100  DDL found, operation [create table
myTableTemp (
    vid varchar2(100),
    someDate date,
    primary key (vid)
) ], start SCN [1186777], commit SCN [1186795] instance [test10g (1)], DDL
seqno [4137].

2011-01-20 15:11:43  GGS INFO    2100  DDL operation excluded [EXCLUDE OBJNAME
myTableTemp OPTYPE CREATE], optype [CREATE], objtype [TABLE], objname
[QATEST1.MYTABLETEMP].
```

Replicat DDL reporting

The Replicat report lists:

- The entire syntax and source Oracle GoldenGate SCN of each DDL operation that Replicat processed from the trail. You can use the source SCN for tracking purposes, especially when there are restores from backup and Replicat is positioned backward in the trail.
- A subsequent entry that shows the scope of the operation (MAPPED, UNMAPPED, OTHER) and how object names were mapped in the target DDL statement, if applicable.
- Another entry that shows how processing criteria was applied.
- Additional entries that show whether the operation succeeded or failed, and whether or not Replicat applied error handling rules.

The following excerpt from a Replicat report illustrates a sequence of steps, including error handling:

```
2011-01-20 15:11:45  GGS INFO     2104  DDL found, operation [drop table
myTableTemp ], Source SCN [1186713.0].

2011-01-20 15:11:45  GGS INFO     2100  DDL is of mapped scope, after mapping
new operation [drop table "QATEST2"."MYTABLETEMP" ].

2011-01-20 15:11:45  GGS INFO     2100  DDL operation included [include objname
myTable*], optype [DROP], objtype [TABLE], objname [QATEST2.MYTABLETEMP].

2011-01-20 15:11:45  GGS INFO     2100  Executing DDL operation.

2011-01-20 15:11:48  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [1].

2011-01-20 15:11:48  GGS INFO     2100  Executing DDL operation , trying again
due to RETRYOP parameter.

2011-01-20 15:11:51  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [2].

2011-01-20 15:11:51  GGS INFO     2100  Executing DDL operation, trying again
due to RETRYOP parameter.

2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [3].

2011-01-20 15:11:54  GGS INFO     2100  Executing DDL operation, trying again
due to RETRYOP parameter.

2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored: error code [942],
filter [include objname myTableTemp], error text [ORA-00942: table or view does
not exist].
```

Statistics in the process reports

You can send current statistics for DDL processing to the Extract and Replicat reports by using the SEND command in GGSCI.

```
SEND {EXTRACT | REPLICAT} <group> REPORT
```

The statistics show totals for:

- All DDL operations
- Operations that are MAPPED in scope
- Operations that are UNMAPPED in scope
- Operations that are OTHER in scope
- Operations that were excluded (number of operations minus included ones)
- Errors (Replicat only)
- Retried errors (Replicat only)
- Discarded errors (Replicat only)
- Ignored operations (Replicat only)

```
From Table QATEST1.MYTABLE:
       #                  inserts:       100
       #                  updates:         0
       #                  deletes:         0
       #                 discards:         0

DDL replication statistics:

                   Operations:        18
            Mapped operations:         4
          Unmapped operations:         0
           Default operations:         0
          Excluded operations:         0
```

Viewing metadata in the DDL history table

Use the DUMPDDL command in GGSCI to view the information that is contained in the DDL history table. This information is stored in proprietary format, but you can export it in human-readable form to the screen or to a series of SQL tables that you can query. The information in the DDL history table is the same as that used by the Extract process.

Because the history data comes from the DDL *before* trigger, it reflects the state of an object before a DDL change. Consequently, there is no data for CREATE operations.

DUMPDDL dumps approximately the first 4000 bytes of each record from the DDL history table. To filter the output, use SQL queries or search redirected standard output.

The format of the metadata is string based. It is fully escaped and supports non-standard characters (such as =, ?, *) in object or column names.

To view DDL history with DUMPDDL

1. Run GGSCI.

2. In GGSCI, log into the database as the owner of the history table.

```
DBLOGIN USERID <user>[, PASSWORD <password>]
```

3. Issue the DUMPDDL command.

```
DUMPDDL [SHOW]
```

Basic DUMPDDL

The basic DUMPDDL command sends metadata to the tables listed in the following table. All of these tables are owned by the Oracle GoldenGate DDL schema that was assigned during the installation of the DDL objects (see the Oracle GoldenGate *Oracle Installation and Setup Guide*). To view the structure of these tables, use the DESC command in SQL*Plus.

Table 21 DUMPDDL output tables

Table	Description
GGS_DDL_OBJECTS	Contains information about the objects in the DDL operations. SEQNO is the primary key. All of the other GGS_DDL_ tables contain a SEQNO column that is the foreign key to GGS_DDL_OBJECTS.
GGS_DDL_COLUMNS	Contains information about the columns of the objects in the DDL operations.
GGS_DDL_LOG_GROUPS	Contains information about the supplemental log groups of the objects in the DDL operations.
GGS_DDL_PARTITIONS	Contains information about the partitions of the objects in the DDL operations.
GGS_DDL_PRIMARY_KEYS	Contains information about the primary keys of the objects in the DDL operations.

The SEQNO column is the DDL sequence number that is listed in the Extract and Replicat report files. It also can be obtained by querying the DDL history table. The default name of the DDL history table is GGS_DDL_HIST.

DUMPDDL SHOW

DUMPDDL with the SHOW option dumps the information from the history table to the screen in standard output format. No GGS_DDL_ output tables are produced. The command dumps all records in the DDL history table.

Tracing DDL processing

If you open a support case with Oracle GoldenGate Technical Support, you might be asked to turn on tracing. The following parameters control DDL tracing.

- TLTRACE controls Extract tracing
- TRACE and TRACE2 control Replicat tracing.

These parameters have options to isolate the tracing of DDL from the tracing of DML. For more information, see the Oracle GoldenGate*Windows and UNIX Reference Guide*.

Tracing the DDL trigger

To trace the activity of the Oracle GoldenGate DDL trigger, use the following tools.

- ggs_ddl_trace.log trace file: Oracle GoldenGate creates a trace file in the USER_DUMP_DEST directory of Oracle. On RAC, each node has its own trace file that captures DDL tracing for that node. You can query the trace file as follows:

  ```
  select value from sys.v_$parameter where name = 'user_dump_dest';
  ```

- ddl_tracelevel script: Edit and run this script to set the trace level. A value of None generates no DDL tracing, except for fatal errors and installation logging. The default value of 0 generates minimal tracing information. A value of 1 or 2 generates a much larger amount of information in the trace file. Do not use 1 or 2 unless requested to do so by a Oracle GoldenGate Technical Support analyst as part of a support case.

- ddl_cleartrace script: Run this script on a regular schedule to prevent the trace file from consuming excessive disk space as it expands. It deletes the file, but Oracle GoldenGate will create another one. The DDL trigger stops writing to the trace file when the Oracle directory gets low on space, and then resumes writing when space is available again. This script is in the Oracle GoldenGate directory. Back up the trace file before running the script.

CHAPTER 15
Configuring DDL synchronization for a Teradata database

About this documentation

This documentation contains information that is specific to the setup of the Oracle GoldenGate solution within a Teradata environment. It assumes that the reader has a fundamental knowledge of the Teradata database and the Teradata Replication Solutions. It also assumes that the following have been configured properly:

- Relay Services Gateway (RSG)
- Change Data Capture (CDC)
- Teradata Access Module (TAM)
- Replication groups

For a complete description of how to configure replication for the Teradata database, see the Teradata Replication Solutions documentation.

Overview of DDL synchronization

Oracle GoldenGate supports the synchronization of DDL operations from one database to another. DDL synchronization can be active when:

- business applications are actively accessing and updating the source and target objects.
- Oracle GoldenGate transactional data synchronization is active.

The components that support the replication of DDL and the replication of transactional data changes (DML) are independent of each other. Therefore, you can synchronize:

- just DDL changes
- just DML changes
- both DDL and DML

For example, if you use batch runs to keep the target objects current, you can configure DDL synchronization as a continuous (online) run, so that the target metadata is always up-to-date when the batch loads are performed. The Oracle GoldenGate batch load will use the current metadata from the source and target catalogs.

For a list of supported objects and operations for DDL support for Teradata, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.

Limitations of Oracle GoldenGate DDL support

DDL statement length

Oracle GoldenGate measures the length of a DDL statement in bytes, not in characters. The supported length is approximately 2 MB, allowing for some internal overhead that can vary in size depending on the name of the affected object and its DDL type, among other characteristics. If the DDL is longer than the supported size, Extract will issue a warning and ignore the DDL operation.

System configuration

- Oracle GoldenGate supports DDL replication in uni-directional configurations and also in bi-directional configurations (active-passive and active-active) between two, and only two, systems.
- Oracle GoldenGate supports DDL synchronization only in a like-to-like configuration. Oracle GoldenGate DDL support requires the following:
  - Source and target object definitions must be identical.
  - The ASSUMETARGETDEFS parameter must be used in the Replicat parameter file. Replicat will abend if objects are configured for DDL support and the SOURCEDEFS parameter is being used. For more information about ASSUMETARGETDEFS, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Filtering, mapping, and transformation

DDL

DDL operations cannot be transformed by any Oracle GoldenGate process. You can use simple string substitution, and you can map schema and object names as follows:

- Source DDL can be mapped and filtered to a different target object by a primary Extract or a Replicat process, but the mapping or filtering of DDL by a data-pump Extract or a VAM-sort Extract is not permitted. DDL is propagated through a data pump or VAM-sort Extract in PASSTHRU mode (non-mapped or filtered).
- As a result, DDL that is performed on a source table of a certain name (for example ALTER TABLE TableA...) will be processed by the data pump or VAM-sort Extract with the same table name (ALTER TABLE TableA). It cannot be mapped by that process as ALTER TABLE TableB, regardless of any TA BLE statements that specify otherwise.

DML

Because DDL is passed through unchanged and unmapped by a data pump or VAM-sort Extract, consider this limitation if you intend to perform DML manipulation that maps source tables to different target names. Perform filtering, mapping, and transformation of DML with the primary Extract or with Replicat, and configure those tables for PASSTHRU mode in the data pump or VAM-sort Extract.

If a data pump or VAM-sort Extract must perform DML manipulation that includes name mapping, then you must also configure Replicat to perform the corresponding name mapping for replicated DDL for those tables.

NOTE Tables that do not use DDL support can be configured in NOPASSTHRU mode to allow data filtering, and manipulation by the data pump.

To configure a table for data pass-through

1. In the parameter file of the data pump or VAM-sort Extract, place the PASSTHRU parameter before all of the TABLE statements that contain tables that use DDL support.
2. In the same parameter file, you can place the NOPASSTHRU parameter before any TABLE statements that contain tables that do not use DDL support, if you want data filtering, mapping, or transformation to be performed for them.
3. Do not use any of the DDL configuration parameters for a data pump or VAM-sort Extract: DDL, DDLOPTIONS, DDLSUBST, DDLERROR, or any of the Oracle GoldenGate tracing parameters with DDL options.

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

SQLEXEC

- 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.

User Exits

Use the GET_DDL_RECORD_PROPERTIES function to return a DDL operation, including information about the object on which the DDL was performed and also the text of the DDL statement itself. The Extract process can only get the source table layout. The Replicat process can get source or target layouts.

This user exit only provides retrieval capability. Oracle GoldenGate does not provide a function for manipulating DDL records.

DDL response time

The response time for DDL statements that are captured for replication might increase because of the inherent latency of the synchronization protocol between the Teradata database and the replication system, including the Oracle GoldenGate component. The response time overhead should not exceed one second under most conditions. The response time of DDL that is not captured should not be significantly affected. The performance cost of capturing changed data in tables with UDTs or LOBs, compared to tables without those data types, should be comparable to other methods of exporting data.

Configuration guidelines for DDL support

Database privileges

See the Oracle GoldenGate *Oracle Installation and Setup Guide* for database privileges that are required for Oracle GoldenGate to support DDL capture and replication.

Initial synchronization

- To configure DDL replication, start with a target database that is synchronized with the source database. DDL support is compatible with the Replicat initial load method.
- DDL support is also compatible with the Teradata FastLoad initial synchronization method. See the Teradata documentation for more information about this feature.
- 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.

Process topology

- If using parallel Extract and/or Replicat processes, keep related DDL and DML together in the same process stream to ensure data integrity. Configure the processes so that:
  - all DDL and DML for any given object are processed by the same Extract group and by the same Replicat group.
  - all objects that are relational to one another are processed by the same process group.

  For example, if ReplicatA processes DML for Table1, then it should also process the DDL for Table1. If Table2 has a foreign key to Table1, then its DML and DDL operations also should be processed by ReplicatA.
- If an Extract group writes to multiple trails that are read by different Replicat groups, Extract sends all of the DDL to all of the trails. Use each Replicat group to filter the DDL by using the filter options of the DDL parameter in the Replicat parameter file.

Object names

- Oracle GoldenGate supports object names that contain multibyte characters and special alphanumeric characters, such as !, $, and #. Limitations apply when objects are mapped with TABLE or MAP parameters, because those parameters do not support all of the possible special characters. DDL on objects in MAP and TABLE statements inherit the limitations of those parameters. For more information about these parameters, see the documentation for MAP and TABLE in the Oracle GoldenGate *Windows and UNIX Reference Guide*.
- You can use standard Oracle GoldenGate asterisk wildcards (*) to specify object names in configuration parameters that support DDL synchronization. To process wildcards correctly, the WILDCARDRESOLVE parameter is set to DYNAMIC by default. If WILDCARDRESOLVE is set to anything else, the Oracle GoldenGate process that is processing DDL operations will abend and write the error to the process report.

- Oracle GoldenGate supports the use of a space before, after, or both before and after, the dot that separates owner and object names in a DDL statement. Only one space on each side of the dot is supported. For example, the following are valid:

```
CREATE TABLE fin . customers...
CREATE TABLE fin. customers...
CREATE TABLE fin .customers...
```

Understanding DDL scopes

Database objects are classified into *scopes*. A scope is a category that defines how DDL operations on an object are handled by Oracle GoldenGate. The scopes are:

- MAPPED
- UNMAPPED
- OTHER

The use of scopes enables granular control over the filtering of DDL operations, string substitutions, and error handling.

Mapped scope

Objects that are specified in TABLE and MAP statements are of *MAPPED scope*. Extraction and replication instructions in those statements apply to both data (DML) and DDL on the specified objects, unless override rules are applied.

For objects in TABLE and MAP statements, the DDL operations listed in the following table are supported.

Table 22 Objects that can be mapped in MAP and TABLE statements

Operations	**Object**[1]
CREATE ALTER DROP RENAME COMMENT ON[2]	TABLE (includes AS SELECT) INDEX[3] TRIGGER VIEW FUNCTION PROCEDUREMACRO
GRANT REVOKE	TABLE

[1] TABLE and MAP do not support some special characters that could be used in an object name affected by these operations. For a list of those characters, see the MAP and TABLE parameter descriptions in the Oracle GoldenGate *Windows and UNIX Reference Guide*. Objects with non-supported special characters are supported by the scopes of UN-MAPPED and OTHER.

[2] Applies to COMMENT ON TABLE, COMMENT ON COLUMN.

[3] DDL on an index is not supported for HASH and JOIN operations.

For Extract, MAPPED scope marks an object for DDL capture according to the instructions in the TABLE statement. For Replicat, MAPPED scope marks DDL for replication and maps it to the object specified by the owner and name in the TARGET clause of the MAP statement.

Assume the following TABLE and MAP statements:

Extract (source)	**Replicat (target)**
`TABLE fin.expen;`	`MAP fin.expen, TARGET fin2.expen2;`
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.bak_*;`

Also assume a source DDL statement of:

```
ALTER TABLE fin.expen ADD notes varchar2(100);
```

In this example, because the source table "fin.expen" is in a MAP statement with a TARGET clause that maps to a different owner and table name, the target DDL statement becomes:

```
ALTER TABLE fin2.expen2 ADD notes varchar2(100);
```

Likewise, the following source and target DDL statements are possible for the second set of TABLE and MAP statements in the example:

Source:	`CREATE TABLE hr.tabPayables ... ;`
Target:	`CREATE TABLE hrBackup.bak_tabPayables ...;`

When objects are of MAPPED scope, you can omit their names from the DDL configuration parameters, unless you want to refine their DDL support further. If you ever need to change the object names in TABLE and MAP statements, the changes will apply automatically to the DDL on those objects.

If you include an object in a TABLE statement, but not in a MAP statement, the DDL for that object is MAPPED in scope on the source but UNMAPPED in scope on the target.

Mapping ALTER INDEX

An ALTER INDEX...RENAME command cannot be mapped to a different target index name, but it can be mapped to a different target owner.

Valid example:

```
ALTER INDEX src.ind RENAME TO indnew;
```

This DDL can be mapped with wildcards as:

```
MAP src.* TARGET tgt.*;
```

Alternatively, it can be mapped explicitly as the following, making sure to use the original index name in the source and target specifications:

```
MAP src.ind TARGET tgt.ind;
```

In either of the preceding cases, the target DDL will be:

```
ALTER INDEX tgt.ind RENAME TO indnew;
```

Invalid example:

A MAP statement such as the following is not valid:

```
MAP src.ind TARGET tgt.indnew;
```

That statement maps the old name to the new name, and the target DDL will become:

```
ALTER INDEX tgt.indnew RENAME TO indnew;
```

Unmapped scope

If a DDL operation is supported for use in a TABLE or MAP statement, but its base object name is not included in one of those parameters, it is of *UNMAPPED scope*.

An object name can be of UNMAPPED scope on the source (not in an Extract TABLE statement), but of MAPPED scope on the target (in a Replicat MAP statement), or the other way around. When Teradata DDL is of UNMAPPED scope in the Replicat configuration, it is applied to the target in one of these ways:

- If the required Replicat connection parameter TARGETDB contains just a DSN (as in tdtarg), but not a database name, it is applied to the target object with the same owner (database name) and object name as in the source DDL.
- If a specific database name is used in TARGETDB (as in db@tdtarg), all of the DDL operations are applied to the target with the owner from TARGETDB.

Other scope

DDL operations that cannot be mapped are of *OTHER scope*. When DDL is of OTHER scope in the Replicat configuration, it is applied to the target with the same owner and object name as in the source DDL.

An example of OTHER scope is a DDL operation that makes a system-specific reference, such as DDL that operates on data file names.

Enabling DDL support

By default, the status of DDL replication support is as follows:

- On the source, Oracle GoldenGate DDL support is disabled by default, although the Teradata TAM sends all of the DDL to the Oracle GoldenGate VAM. You must configure Extract to capture DDL by using the DDL parameter.
- On the target, DDL support is enabled by default, to maintain the integrity of transactional data that is replicated. By default, Replicat will process all DDL operations that the trail contains. If needed, you can use the DDL parameter to configure Replicat to ignore or filter DDL operations.

Filtering DDL replication

Use the DDL parameter to filter DDL operations so that specific (or all) DDL is applied to the target database according to your requirements.

When used without options, the DDL parameter performs no filtering, and it causes all DDL operations to be propagated as follows:

- As an Extract parameter, it captures all supported DDL operations that are generated on all supported database objects and sends them to the trail.

- As a Replicat parameter, it replicates all DDL operations from the Oracle GoldenGate trail and applies them to the target. This is the same as the default behavior without this parameter.

When used with options, the DDL parameter acts as a filtering agent to include or exclude DDL operations based on:

- scope
- object type
- operation type
- object name
- strings in the DDL command syntax or comments, or both

Only one DDL parameter can be used in a parameter file, but you can combine multiple inclusion and exclusion options to filter the DDL to the required level.

- DDL filtering options are valid for a primary Extract that captures from the transaction source, but not for a data-pump Extract.
- When combined, multiple filter option specifications are linked logically as "AND" statements.
- All filter criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex DDL filtering criteria, it is recommended that you test your configuration in a test environment before using it in production.

NOTE Before you create a DDL parameter statement, it might help to review "How DDL is evaluated for processing" in this chapter.

Syntax

```
DDL [
{INCLUDE | EXCLUDE}
    [, MAPPED | UNMAPPED | OTHER | ALL]
    [, OPTYPE <type>]
    [, OBJTYPE '<type>']
    [, OBJNAME "<name>"]
    [, INSTR '<string>']
]
[...]
```

Table 23 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME "hr.*"`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME "hr.*"`
	`DDL INCLUDE OBJNAME "fin.*" EXCLUDE "fin.ss"`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

Table 23 DDL inclusion and exclusion options

Option	Description

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	
	In the following example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the triggeror index.
	For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

Table 23 DDL inclusion and exclusion options

Option	Description
`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

Combining DDL parameter options

The following is an example of how to combine DDL parameter options.

```
DDL  &
INCLUDE UNMAPPED &
   OPTYPE alter &
   OBJTYPE 'table' &
   OBJNAME "users.tab*" &
INCLUDE MAPPED OBJNAME "*" &
EXCLUDE MAPPED OBJNAME "temporary.tab*"
```

The combined filter criteria in this statement specify the following:

- INCLUDE all ALTER TABLE statements for tables that are not mapped with a TABLE or MAP statement (UNMAPPED scope),
  - only if those tables are owned by "users" and their names start with "tab,"
- and INCLUDE all DDL operation types for all tables that are mapped with a TABLE or MAP statement (MAPPED scope).

- and EXCLUDE all DDL operation types for all tables that are MAPPED in scope,
  - only if those tables are owned by "temporary."
  - and only if their names begin with "tab."

DDL EXCLUDE ALL

DDL EXCLUDE ALL is a special processing option that maintains up-to-date object metadata for Oracle GoldenGate, while blocking the replication of the DDL operations themselves. You can use DDL EXCLUDE ALL when using a method other than Oracle GoldenGate to apply DDL to the target, but you want Oracle GoldenGate to replicate data changes to the target objects. It provides the current metadata to Oracle GoldenGate as objects change, thus preventing the need to stop and start the Oracle GoldenGate processes. The following special conditions apply to DDL EXCLUDE ALL:

- DDL EXCLUDE ALL does not require the use of an INCLUDE clause.
- When using DDL EXCLUDE ALL, you may set the WILDCARDRESOLVE parameter to IMMEDIATE to allow immediate DML resolution if required.

To prevent all DDL metadata and operations from being replicated, omit the DDL parameter entirely.

How Oracle GoldenGate handles derived object names

DDL operations can contain a *base object* name and also a *derived object* name. A base object is an object that contains data. A derived object is an object that inherits some attributes of the base object to perform a function related to that object. DDL statements that have both base and derived objects are:

- RENAME
- CREATE and DROP on an index or trigger

Consider the following DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

In this case, the table is the base object. Its name (hr.tabPayroll) is the *base name* and is subject to mapping with TABLE or MAP under the MAPPED scope. The derived object is the index, and its name (hr.indexPayrollDate) is the *derived name*.

You can map a derived name in its own TABLE or MAP statement, separately from that of the base object. Or, you can use one MAP statement to handle both. In the case of MAP, the conversion of derived object names on the target works as follows.

MAP exists for base object, but not derived object

If there is a MAP statement for the base object, but not for the derived object, the result is an *implicit mapping* of the derived object. Assuming the DDL statement includes MAPPED, Replicat gives the derived object the same target owner as that of the base object. The name of the derived object stays the same as in the source statement. For example, assume the following:

Extract (source)	**Replicat (target)**
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.*;`

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The CREATE INDEX statement is executed by Replicat on the target as:

```
CREATE INDEX hrBackup.indexPayrollDate ON TABLE hrBackup.tabPayroll
(payDate);
```

The rule for the implicit mapping is based the typical industry practice of giving derived objects the same owner as the base object. Also, when indexes are owned by the same target owner as the base object, an implicit mapping eliminates the need to map derived object names explicitly.

MAP exists for base and derived objects

If there is a MAP statement for the base object and also one for the derived object, the result is an *explicit mapping*. Assuming the DDL statement includes MAPPED, Replicat converts the owner and name of each object according to its own TARGET clause. For example, assume the following:

Extract (source)	**Replicat (target)**
`TABLE hr.tab*;`	`MAP hr.tab*, TARGET hrBackup.*;`
`TABLE hr.index*;`	`MAP hr.index*, TARGET hrIndex.*;`

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The CREATE INDEX statement is executed by Replicat on the target as:

```
CREATE INDEX hrIndex.indexPayrollDate ON TABLE hrBackup.tabPayroll
(payDate);
```

Use an explicit mapping when the index on the target must be owned by a different owner from that of the base object, or when the name on the target must be different from that of the source.

MAP exists for derived object, but not base object

If there is a MAP statement for the derived object, but not for the base object, Replicat does not perform any name conversion for either object. The target DDL statement is the same as that of the source. To map a derived object, the choices are:

- Use an explicit MAP statement for the base object.
- If names permit, map both base and derived objects in the same MAP statement by means of a wildcard.
- Create a MAP statement for each object, depending on how you want the names converted.

New tables as derived objects

The following explains how Oracle GoldenGate handles new tables that are created from:

- RENAME
- CREATE TABLE AS SELECT

RENAME

In RENAME operations, the base object is always the new table name. In the following example, the base object name is considered to be "index_paydate."

```
RENAME hr.indexPayrollDate TO index_paydate;
```

The derived object name is "hr.indexPayrollDate."

See "Controlling whether renames enter the DDL configuration" on page 204 for additional information on renames as they relate to DDL replication.

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT statements include SELECT statements and INSERT statements that affect any number of underlying objects. On the target, Oracle GoldenGate obtains the data for the AS SELECT clause from the target database. The objects in the AS SELECT clause must exist in the target database, and their names must be identical to the ones on the source.

In a MAP statement, Oracle GoldenGate only maps the name of the new table (CREATE TABLE <name>) to the TARGET specification, but does not map the names of the underlying objects from the AS SELECT clause. There could be dependencies on those objects that could cause data inconsistencies if the names were converted to the TARGET specification.

The following shows an example of a CREATE TABLE AS SELECT statement on the source and how it would be replicated to the target by Oracle GoldenGate.

```
CREATE TABLE a.tab1 AS SELECT * FROM a.tab2;
```

The MAP statement for Replicat is:

```
MAP a.tab*, TARGET a.x*;
```

The target DDL statement that is applied by Replicat is this:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.tab2;
```

not this:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.xtab2;
```

The name of the table in the AS SELECT * FROM clause remains as it was on the source: tab2.

To keep the data in the underlying objects consistent on source and target, you can configure them for data replication by Oracle GoldenGate. In the preceding example, you could use the following statements to accommodate this requirement:

Source	Target

`TABLE a.tab*;`	`MAPEXCLUDE a.tab2`
	`MAP a.tab*, TARGET a.x*;`
	`MAP a.tab2, TARGET a.tab2;`

Disabling the mapping of derived objects

Use the DDLOPTIONS parameter with the NOMAPDERIVED option to prevent the conversion of the name of a derived object according to a TARGET clause of a MAP statement that includes it. NOMAPDERIVED overrides any explicit MAP statements that contain the name of the base or derived object. Source DDL that contains derived objects is replicated to the target with the same owner and object names as on the source.

The following table shows the results of MAPDERIVED compared to NOMAPDERIVED, based on whether there is a MAP statement just for the base object, just for the derived object, or for both.

Table 24 [NO]MAPDERIVED results on target based on mapping configuration

Base Object	Derived Object	MAP/NOMAP DERIVED?	Derived object converted per a MAP?	Derived object gets owner of base object?
mapped[1]	mapped	MAPDERIVED	yes	no
mapped	not mapped	MAPDERIVED	no	yes
not mapped	mapped	MAPDERIVED	no	no
not mapped	not mapped	MAPDERIVED	no	no
mapped	mapped	NOMAPDERIVED	no	no
mapped	not mapped	NOMAPDERIVED	no	no
not mapped	mapped	NOMAPDERIVED	no	no
not mapped	not mapped	NOMAPDERIVED	no	no

[1] Mapped means included in a MAP statement.

The following examples illustrate the results of MAPDERIVED as compared to NOMAPDERIVED.

In the following table, both trigger and table are owned by "rpt" on the target because both base and derived names are converted by means of MAPDERIVED.

Table 25 Default mapping of derived object names (MAPDERIVED)

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER rpt.act_trig ON rpt.acct;`

In the following table, the trigger is owned by "fin," because conversion is prevented by means of NOMAPDERIVED.

Table 26 Mapping of derived object names when using NOMAPDERIVED

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER fin.act_trig ON rpt.acct;`

NOTE In the case of a RENAME statement, the new table name is considered to be the base table name, and the old table name is considered to be the derived table name.

Using DDL string substitution

You can substitute strings within a DDL operation while it is being processed by Oracle GoldenGate. This feature provides a convenience for changing and mapping directory names and other things that are not directly related to data structures. String substitution is controlled by the DDLSUBST parameter.

Guidelines for using DDLSUBST

- Do not use DDLSUBST to convert column names and data types to something different on the target. Changing the structure of a target object in this manner will cause errors when data is replicated to it. Likewise, do not use DDLSUBST to change owner and table names in a target DDL statement. Always use a MAP statement to map a replicated DDL operation to a different target object.
- DDLSUBST always executes after the DDL parameter, regardless of their relative order in the parameter file. Because the filtering executes first, use filtering criteria that is compatible with the criteria that you are using for string substitution. For example, consider the following parameter statements:

  ```
  DDL INCLUDE OBJNAME "fin.*"
  DDLSUBST 'cust' WITH 'customers' INCLUDE OBJNAME "sales.*"
  ```

  In this example, no substitution occurs because the objects in the INCLUDE and DDLSUBST statements are different. The fin-owned objects are included in the Oracle GoldenGate DDL configuration, but the sales-owned objects are not.
- You can use multiple DDLSUBST parameters. They execute in the order listed in the parameter file.
- There is no maximum string size for substitutions, other than the limit that is imposed by the database. If the string size exceeds the database limit, the Extract or Replicat process that is executing the operation abends.

NOTE Before you create a DDLSUBST parameter statement, it might help to review "How DDL is evaluated for processing" in this chapter.

Syntax

```
DDLSUBST '<search_string>' WITH '<replace_string>'
[INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>]
```

Argument	Description
`'<search_string>'`	The string in the source DDL statement that you want to replace. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.
`WITH`	Required keyword.
`'<replace_string>'`	The string that you want to use as the replacement in the target DDL. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.

`INCLUDE <inclusion clause> |`
`EXCLUDE <exclusion clause>`
	Use one or more INCLUDE and EXCLUDE statements to filter the DDL operations for which the string substitution rules are applied. See the following table.

Table 27 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME "hr.*"`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME "hr.*"`
	`DDL INCLUDE OBJNAME "fin.*" EXCLUDE "fin.ss"`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

Table 27 DDL inclusion and exclusion options

Option	Description
`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`

Table 27 DDL inclusion and exclusion options

Option	Description

	In the following example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the triggeror index.
	For example, to include the following DDL statement, the correct value is “hr.accounts,” not “hr.insert_trig.”
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is “hr.acct.”
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

Example In the following example, the string 'cust' is replaced with the string 'customers' for tables owned by "fin".

```
DDLSUBST 'cust' WITH 'customers'
INCLUDE ALL OBJTYPE 'table' OBJNAME "fin.*"
```

The search is not case-sensitive. To represent a quotation mark in a string, use a double quotation mark.

Example This example uses multiple DDLSUBST parameters. They will execute in the order listed in the parameter file. The net effect is to substitute "a" and "b" strings with "c."

```
DDLSUBST 'a' WITH 'b' INCLUDE ALL
DDLSUBST 'b' WITH 'c' INCLUDE ALL
```

Controlling whether renames enter the DDL configuration

You can use the DDLOPTIONS parameter with the NOCROSSRENAME option to enforce the rule that objects which are excluded from the Oracle GoldenGate configuration cannot be renamed to names that are in the configuration. This is an example of how a rename could occur:

- TableA is excluded, but tableB is included.
- TableA gets renamed to tableB.

If an object does get renamed to one that is in the Oracle GoldenGate configuration, Extract issues a warning, so that you can take the appropriate action (keep it in the Oracle GoldenGate configuration or make the appropriate parameter adjustments to exclude it). An example of how this notification is useful is to prevent errors if a renamed object has a structure that is not supported by Oracle GoldenGate.

NOCROSSRENAME applies globally to:

- all objects specified in TABLE and TABLEEXCLUDE statements in the parameter file
- all objects that are excluded from the Oracle GoldenGate DDL configuration, and not specified by TABLE or TABLEEXCLUDE.

DDLOPTIONS NOCROSSRENAME provides the same results as the TABLEEXCLUDE parameter, when used with the NORENAME option. The difference between the two parameters is that TABLEEXCLUDE NORENAME allows more selectivity than NOCROSSRENAME, because the functionality only affects the objects in the TABLEEXCLUDE statement.

How DDL is evaluated for processing

The following explains how Oracle GoldenGate processes DDL statements on the source and target systems. It shows the order in which different criteria in the Oracle GoldenGate parameters are processed, and it explains the differences between how Extract and Replicat each process the DDL.

Extract

1. Extract captures a DDL operation.

2. Extract searches for the DDL parameter. (This example assumes it exists.)
3. Extract gets the base object name and, if present, the derived object name.
4. Extract determines the DDL scope: MAPPED, UNMAPPED or OTHER. It is MAPPED if:
   - the operation and object types are supported for mapping.

     and...
   - the base object name and/or derived object name (if RENAME) is in a TABLE parameter.

   It is UNMAPPED if:
   - the operation and object types are not supported for mapping.

     and...
   - the base object name and/or derived object name (if RENAME) is not in a TABLE parameter.

   Otherwise the operation is identified as OTHER.
5. Extract checks the DDL parameter for INCLUDE and EXCLUDE clauses, and it evaluates the DDL parameter criteria in those clauses. All options must evaluate to TRUE in order for the INCLUDE or EXCLUDE to evaluate to TRUE. The following occurs:
   - If an EXCLUDE clause evaluates to TRUE, Extract discards the DDL operation and evaluates another DDL operation. In this case, the processing steps start over.
   - If an INCLUDE clause evaluates to TRUE, or if the DDL parameter does not have any INCLUDE or EXCLUDE clauses, Extract includes the DDL operation, and the processing logic continues.
6. Extract searches for a DDLSUBST parameter and evaluates the INCLUDE and EXCLUDE clauses. If the criteria in those clauses add up to TRUE, Extract performs string substitution. Extract evaluates the DDL operation against each DDLSUBST statement in the parameter file. For all true DDLSUBST statements, Extract performs string substitution in the order that the DDLSUBST parameters are listed in the file.
7. Extract writes the DDL statement to the trail.

Replicat

1. Replicat reads the DDL operation from the trail.
2. Replicat evaluates the DDL synchronization scope to determine if the DDL qualifies for name mapping. Anything else is of OTHER scope.
3. Replicat evaluates the MAP statements in the parameter file. If the source base object name for this DDL (as read from the trail) appears in any of the MAP statements, the operation is marked as MAPPED in scope. Otherwise it is marked as UNMAPPED in scope.
4. Replicat replaces the source base object name with the base object name that is specified in the TARGET clause of the MAP statement.
5. If there is a derived object, Replicat searches for DDLOPTIONS MAPDERIVED. If it is present, Replicat replaces the source derived name with the target derived name from the MAP statement.

6. Replicat checks the DDL parameter for INCLUDE and EXCLUDE clauses, and it evaluates the DDL parameter criteria contained in them. All options must evaluate to TRUE in order for the INCLUDE or EXCLUDE to evaluate to TRUE. The following occurs:
   - If any EXCLUDE clause evaluates to TRUE, Replicat discards the DDL operation and starts evaluating another DDL operation. In this case, the processing steps start over.
   - If any INCLUDE clause evaluates to TRUE, or if the DDL parameter does not have any INCLUDE or EXCLUDE clauses, Replicat includes the DDL operation, and the processing logic continues.
7. Replicat searches for the DDLSUBST parameter and evaluates the INCLUDE and EXCLUDE clauses. If the options in those clauses add up to TRUE, Replicat performs string substitution. Replicat evaluates the DDL operation against each DDLSUBST statement in the parameter file. For all true DDLSUBST statements, Replicat performs string substitution in the order that the DDLSUBST parameters are listed in the file.
8. Replicat executes the DDL operation on the target database.
9. If there are no errors, Replicat processes the next DDL statement. If there are errors, Replicat performs the following steps.
10. Replicat analyzes the INCLUDE and EXCLUDE rules in the Replicat DDLERROR parameter statements in the order that they appear in the parameter file. If Replicat finds a rule for the error code, it applies the specified error handling; otherwise, it applies DEFAULT handling.
11. If the error handling does not enable the DDL operation to succeed, Replicat does one of the following: abends, ignores the operation, or discards it as specified in the rules.

NOTE If there are multiple targets for the same source in a MAP statement, the processing logic executes for each one.

Handling Extract DDL processing errors

Use the Extract option of the DDLERROR parameter to handle errors on objects found by Extract for which metadata cannot be found.

Syntax
```
DDLERROR [RESTARTSKIP <num skips>] [SKIPTRIGGERERROR <num errors>]
```

Where:

- RESTARTSKIP skips a number of DDL operations on startup to prevent Extract from abending on an error. By default, Extract abends on an error so that no operations are skipped. You can skip up to 100,000 DDL operations.

To write information about skipped operations to the Extract report file, use the DDLOPTIONS parameter with the REPORT option.

Handling Replicat DDL processing errors

Use the Replicat options of the DDLERROR parameter to handle errors that occur when DDL is applied to the target database. With DDLERROR options, you can handle most errors in a default manner, for example to stop processing, and also handle other errors in a specific manner. You can use multiple instances of DDLERROR in the same parameter file to handle all errors that are anticipated.

Use the combination of <error>, DEFAULT, and <response> to create rules for how Replicat responds to anticipated and unanticipated DDL errors. Make certain to specify the appropriate inclusion and exclusion clauses to apply the rules to the intended DDL. Then, use additional options to refine the error handling, as needed.

Syntax

```
DDLERROR
{<error> | DEFAULT} {<response>}
{INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>}
[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]
```

Argument	Description

`{<error> | DEFAULT} {<response>}`	◆ <error> is a specific DDL error that you want to be handled with this statement.
	◆ DEFAULT sets a global response to all DDL errors except those for which explicit DDLERROR statements are specified.
	◆ <response> can be one of the following:
	
	`ABEND`
	Rolls back the operation and terminates processing abnormally. ABEND is the default.
	
	`DISCARD`
	Logs the offending operation to the discard file but continue processing subsequent DDL. Specify a discard file with the DISCARDFILE parameter.
	
	`IGNORE`
	Ignores the error.
	
	`RETRYOP MAXRETRIES <n> [RETRYDELAY <delay>]`
	Retries the offending operation. Use the MAXRETRIES option to control the number of retries. Replicat abends after the specified number of MAXRETRIES. Specify a whole integer.
	Use RETRYDELAY to set the amount of time, in seconds, between retry attempts.

`{INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>}`	Controls whether specific DDL is handled or not handled by the DDLERROR statement. See the following table for descriptions.

Argument	Description
`[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]`	Controls whether or not Extract abends when DML is issued on objects that could not be found on the target. This condition is typically caused by DDL that is issued directly on the target outside of replication, or by a discrepancy between source and target definitions. IGNOREMISSINGOBJECTS causes Replicat to skip DML operations on missing tables. ABENDONMISSINGOBJECTS causes Replicat to abend on DML operations on missing tables.

Table 28 DDL inclusion and exclusion options

Option	Description

`INCLUDE | EXCLUDE`	Use INCLUDE and EXCLUDE to identify the beginning of an inclusion or exclusion clause.
	◆ An inclusion clause contains filtering criteria that identifies the DDL that this parameter will affect.
	◆ An exclusion clause contains filtering criteria that excludes specific DDL from this parameter.
	The inclusion or exclusion clause must consist of the INCLUDE or EXCLUDE keyword followed by any valid combination of other options of the parameter that is being applied.
	
	If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:
	`DDL EXCLUDE OBJNAME "hr.*"`
	However, you can use either of the following:
	`DDL INCLUDE ALL, EXCLUDE OBJNAME "hr.*"`
	`DDL INCLUDE OBJNAME "fin.*" EXCLUDE "fin.ss"`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

`MAPPED | UNMAPPED | OTHER | ALL`	Use MAPPED, UNMAPPED, OTHER, and ALL to apply INCLUDE or EXCLUDE based on the DDL operation scope. ◆ MAPPED applies INCLUDE or EXCLUDE to DDL operations that are of MAPPED scope. MAPPED filtering is performed before filtering that is specified with other DDL parameter options. ◆ UNMAPPED applies INCLUDE or EXCLUDE to DDL operations that are of UNMAPPED scope. ◆ OTHER applies INCLUDE or EXCLUDE to DDL operations that are of OTHER scope. ◆ ALL applies INCLUDE or EXCLUDE to DDL operations of all scopes.

Table 28 DDL inclusion and exclusion options

Option	Description
`OPTYPE <type>`	Use OPTYPE to apply INCLUDE or EXCLUDE to a specific type of DDL operation, such as CREATE, ALTER, and RENAME. For <type>, use any DDL command that is valid for the database. For example, to include ALTER operations, the correct syntax is: `DDL INCLUDE OPTYPE ALTER`

`OBJTYPE '<type>'`	Use OBJTYPE to apply INCLUDE or EXCLUDE to a specific type of database object. For <type>, use any object type that is valid for the database, such as TABLE, INDEX, and TRIGGER. For an Oracle materialized view and materialized views log, the correct types are snapshot and snapshot log, respectively. Enclose the name of the object type within single quotes. For example:
	`DDL INCLUDE OBJTYPE 'INDEX'`
	`DDL INCLUDE OBJTYPE 'SNAPSHOT'`
	For Oracle object type USER, do not use the OBJNAME option, because OBJNAME expects "owner.object" whereas USER only has a schema.

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object, for example owner.table_name. This option takes a double-quoted string as input.
	You can use a wildcard only for the object name.
	Example:
	`DDL INCLUDE OBJNAME "accounts.*"`
	Do not use OBJNAME for the Oracle USER object, because OBJNAME expects "owner.object" whereas USER only has a schema.
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	In the following example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.

Table 28 DDL inclusion and exclusion options

Option	Description

	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the triggeror index.
	For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive. INSTR does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

`INSTRWORDS '<word list>'`	Use INSTRWORDS to apply INCLUDE or EXCLUDE to DDL statements that contain the specified words.
	For <word list>, supply the words in any order, within single quotes. To include spaces, put the space (and the word, if applicable) in double quotes. Double quotes also can be used to enclose sentences.
	All specified words must be present in the DDL for INSTRWORDS to take effect.
	Example:
	`ALTER TABLE INCLUDE INSTRWORDS 'ALTER CONSTRAINT " xyz"`
	This example will match
	`ALTER TABLE ADD CONSTRAINT xyz CHECK`
	and
	`ALTER TABLE DROP CONSTRAINT xyz`
	INSTRWORDS does not support single quotation marks (`' '`) that are within the string, nor does it support NULL values.

Sample DDLERROR statement

In the following example, the DDLERROR statement causes Replicat to ignore the specified error, but not before trying the operation again three times at ten-second intervals. Replicat applies the error handling to DDL operations executed on objects whose names satisfy the wildcard of "tab*" (any user, any operation) except those that satisfy "tab1*."

```
DDLERROR <error> IGNORE RETRYOP MAXRETRIES 3 RETRYDELAY 10 &
INCLUDE ALL OBJTYPE TABLE OBJNAME "tab*" EXCLUDE OBJNAME "tab1*"
```

To handle all errors except that error, the following DDLERROR statement can be added.

```
DDLERROR DEFAULT ABENDS
```

In this case, Replicat abends on DDL errors.

Using multiple DDLERROR statements

The order in which you list DDLERROR statements in the parameter file does not affect their validity unless multiple DDLERROR statements specify the same error, without any additional qualifiers. In that case, Replicat only uses the first one listed. For example, given the following statements, Replicat will abend on the error.

```
DDLERROR <error1> ABEND
DDLERROR <error1> IGNORE
```

With the proper qualifiers, however, the previous configuration becomes a more useful one. For example:

```
DDLERROR <error1> ABEND INCLUDE OBJNAME "tab*"
DDLERROR <error1> IGNORE
```

In this case, because there is an INCLUDE statement, Replicat will abend only if an object name in an errant DDL statement matches wildcard "tab*." Replicat will ignore errant operations that include any other object name.

Viewing DDL report information

By default, Oracle GoldenGate shows basic statistics about DDL operations at the end of the Extract and Replicat reports. To enable expanded DDL reporting, use the DDLOPTIONS parameter with the REPORT option. Expanded reporting includes the following information about DDL processing:

- A step-by-step history of the DDL operations that were processed by Oracle GoldenGate.
- The DDL filtering and processing parameters that are being used.

Expanded DDL report information increases the size of the report file, but it might be useful in certain situations, such as for troubleshooting.

To view a process report

To view a report, use the VIEW REPORT command in GGSCI.

```
VIEW REPORT <group>
```

Extract DDL reporting

The Extract report lists the following:

- The entire syntax of each captured DDL operation, its Oracle GoldenGate CSN number, the Teradata sequence number, and the size of the operation in bytes.
- A subsequent entry that shows how processing criteria was applied to the operation, for example string substitution or INCLUDE and EXCLUDE filtering.
- Another entry showing whether the operation was written to the trail or excluded.

The following is an example taken from an Extract report file.

```
2011-01-21 18:41:40  GGS INFO       2100  DDL found, operation [DROP TABLE
"SMIJATOVDBS"."src13_tabtable_9" ; (size 59)], start CSN [2500FF3F0200363A],
DDL seqno [00000025000000000000381500000021].
2011-01-21 18:41:40  GGS INFO      2100  DDL operation included [include mapped
objname "*"], optype [DROP], objtype [TABLE], objowner [SMIJATOVDBS], objname
[SRC13_TABTABLE_9].
2011-01-21 18:41:40  GGS INFO       2100  DDL operation written to extract
trail file.
```

Replicat DDL reporting

The Replicat report lists:

- The entire syntax of each DDL operation that Replicat processed from the trail.
- A subsequent entry that shows the scope of the operation (MAPPED, UNMAPPED, OTHER) and how object names were mapped in the target DDL statement, if applicable.
- Another entry that shows how processing criteria was applied.
- Additional entries that show whether the operation succeeded or failed, and whether or not Replicat applied error handling rules.

The following is an example taken from a Replicat parameter file.

```
2011-01-21 18:41:44  GGS INFO       2104  DDL found, operation [DROP TABLE
"SMIJATOVDBS"."src13_tabtable_9" ; (size 59)].

2011-01-21 18:41:44  GGS INFO      2100  DDL is of mapped scope, after mapping
new operation [DROP TABLE "SMIJATOVDBT"."SRC13_TABTABLE_9" ; (size 59)].

2011-01-21 18:41:44  GGS INFO       2100  Executing DDL operation.


2011-01-21 18:41:44  GGS INFO       2105  DDL operation successful.
```

Statistics in the process reports

You can send current statistics for DDL processing to the Extract and Replicat reports by using the SEND command in GGSCI.

```
SEND {EXTRACT | REPLICAT} <group> REPORT
```

The statistics show totals for:

- All DDL operations
- Operations that are MAPPED in scope
- Operations that are UNMAPPED in scope
- Operations that are OTHER in scope
- Operations that were excluded (number of operations minus included ones)
- Errors (Replicat only)
- Retried errors (Replicat only)
- Discarded errors (Replicat only)
- Ignored operations (Replicat only)

```
From Table QATEST1.MYTABLE:
       #                   inserts:       100
       #                   updates:         0
       #                   deletes:         0
       #                  discards:         0

DDL replication statistics:

                    Operations:        18
             Mapped operations:         4
           Unmapped operations:         0
            Default operations:         0
           Excluded operations:         0
```

Tracing DDL processing

If you open a support case with Oracle GoldenGate Technical Support, you might be asked to turn on tracing. The following parameters control DDL tracing.

- TLTRACE controls Extract tracing
- TRACE and TRACE2 control Replicat tracing.

These parameters have options to isolate the tracing of DDL from the tracing of DML. For more information, see the Oracle GoldenGate*Windows and UNIX Reference Guide*.

CHAPTER 16

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 217. The utility performs the initial load.
- "Loading data from file to Replicat" on page 218. 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 223. 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 228. 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 232. 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 236. 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 141 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 the Manager process" on page 23.

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 11.

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*.

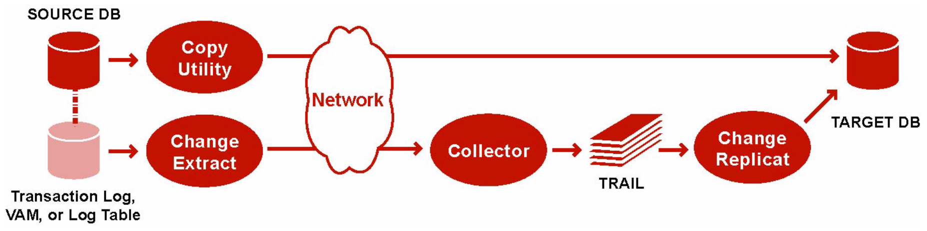
Sharing parameters between process groups

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 33.

For more information about macros, see page 269.

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 215.
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
   ```

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 with the EDIT PARAMS command.

   ```
   VIEW PARAMS <group name>
   EDIT PARAMS <group name>
   ```

Where: <group name> is the name of the Replicat group.

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
```

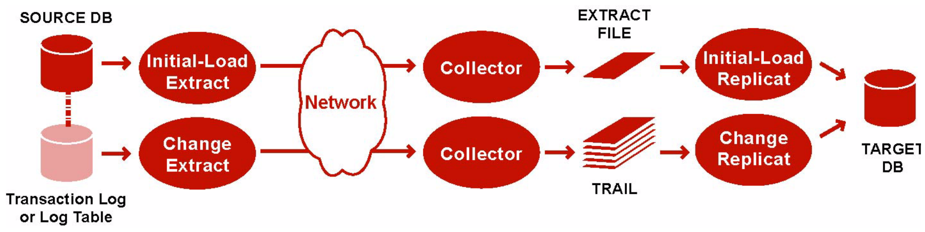
13. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

```
EDIT PARAMS <Replicat group name>
```

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 215.
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 29 in the order shown, starting a new line for each parameter statement.

Table 29 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>]]`
◆ 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 ggs123`
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>`
	Specifies the target system and port where Manager is running.

`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: On Solaris systems, the size of an extract file cannot exceed 2GB if it will be processed by Replicat. Use the MAXFILES and MEGABYTES options to control the size.

Table 29 Initial-load Extract parameters for loading data from file to Replicat (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 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 30 in the order shown, starting a new line for each parameter statement.

Table 30 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.

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. 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 extract file specified with the Extract parameter RMTFILE.

Table 30 Initial-load Replicat parameters for loading data from file to Replicat (continued)

Parameter	Description
`{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 11.

`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
```

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
    ```

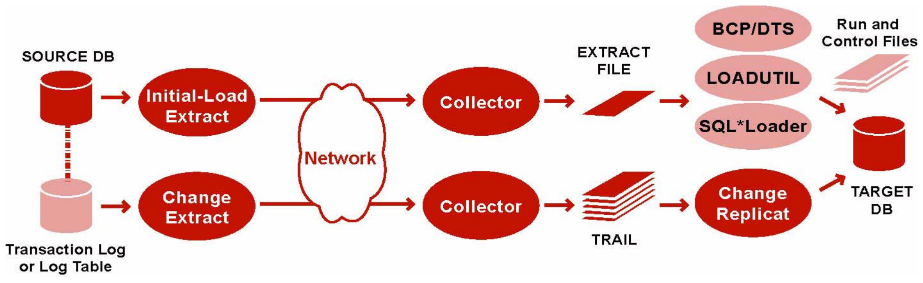
23. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

    ```
    EDIT PARAMS <Replicat group name>
    ```

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 215.
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 31 in the order shown, starting a new line for each parameter statement.

Table 31 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.

Table 31 Initial-load Extract parameters for loading from file to database utility (continued)

Parameter	Description

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
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>`
`[, PARAMS - E -d <defs file>]`
◆ -E converts ASCII to EBCDIC.
◆ -d <defs file> specifies the source definitions file.
	Specifies the target system and port where Manager is running.
	The PARAMS clause is necessary when loading with IBM's Load Utility, because Oracle GoldenGate will need to refer to the source definitions file.

`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.

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

Table 32 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*.

`[TARGETDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
	Specifies database connection information. These parameters also allow for authentication at the operating-system level. 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 11.

Table 32 Initial-load Replicat parameters for loading from file to database utility (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.

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
    ```

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
    ```

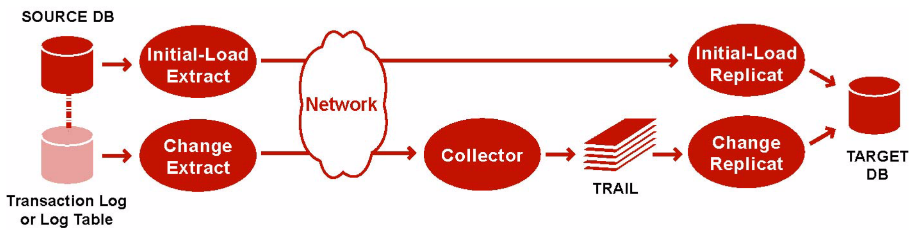
25. On the target system, edit the Replicat parameter file to remove the HANDLECOLLISIONS parameter. This prevents HANDLECOLLISIONS from being enabled again the next time Replicat starts.

    ```
    EDIT PARAMS <Replicat group name>
    ```

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 218 or "Loading data from file to database utility" on page 223.

You can control which port is used by Replicat by specifying the DYNAMICPORTLIST parameter in the Manager parameter file. When starting a process such as Replicat, Manager first looks for a port defined with DYNAMICPORTLIST. If no ports are listed, Manager chooses a port number by incrementing from its own port number until a port is available.

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 215.
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 33 in the order shown, starting a new line for each parameter statement.

Table 33 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.

`[SOURCEDB <dsn>,]`
`[USERID <user id>[, PASSWORD <pw>]]`
◆ 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 ggs123`
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>`
	Specifies the target system and port where Manager is running.

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

Table 34 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>]]`
◆ 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 ggs123`
	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 11.

`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
```

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.

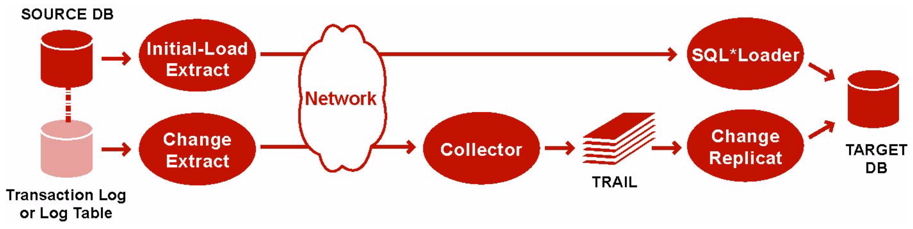
```
SEND REPLICAT <Replicat group name>, NOHANDLECOLLISIONS
```

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.

```
EDIT PARAMS <Replicat group name>
```

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.

You can control which port is used by Replicat by specifying the DYNAMICPORTLIST parameter in the Manager parameter file. When starting a process such as Replicat, Manager first looks for a port defined with DYNAMICPORTLIST. If no ports are listed, Manager chooses a port number by incrementing from its own port number until a port is available.

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 218 or "Loading data from file to database utility" on page 223.
- 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 215.
2. (Oracle 9i and later) 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 35 in the order shown, starting a new line for each parameter statement.

Table 35 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>]]`
◆ 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 ggs123`
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>`
	Specifies the target system and port where Manager is running.

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

Table 36 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>`
	Specifies the user ID and password to be used by the initial-load Replicat for connecting to the Oracle target database. You can include a host string, for example:
	`USERID ggs@ora1.ora,`
	`PASSWORD ggs123`
	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 11.

Table 36 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
    ```

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.

    ```
    EDIT PARAMS <Replicat group name>
    ```

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 12.

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. Edit the Replicat parameter file(s).

    ```
    EDIT PARAMS <Replicat group name>
    ```

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 17
Mapping and manipulating data

Overview of data mapping and manipulation

You can integrate data between different source and target tables by:

- Selecting records and columns
- Selecting and converting operations
- Mapping dissimilar columns
- Using transaction history
- Testing and transforming data
- Using tokens

All data selection, mapping, and manipulation that Oracle GoldenGate performs is accomplished by using 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 and conversion between tables that have different data structures requires either a source-definitions file, a target-definitions file, or in some cases both. For more information about how to create a source-definitions or target-definitions file, see Chapter 11 on page 115.

Limitations of support

- Some Oracle GoldenGate features and functionality do not support the use of data filtering and manipulation. Where applicable, this limitation is documented.
- 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.

Deciding where data mapping and conversion will take place

Column mapping and conversion can be performed on the source system, on the target system, or on an intermediary system. The exception is when the replication is from a Windows or UNIX system to a NonStop target, which always requires those functions to be performed on the source because Replicat for NonStop cannot convert two-part table names and data types to the three-part names that are used on the NonStop platform. The mapping and conversion must be performed on the source Windows or UNIX system, so that Extract can format the trail data with NonStop names and target data types.

To prevent added overhead on the source system, most Oracle GoldenGate users choose to perform 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.

Handling anomalies in data from NonStop systems

When moving data between a Windows or UNIX system and a NonStop Server source or target, certain anomalies can be caused by some applications running on NonStop Server in which transactional operations are sent out of order to the TMF audit trail, and subsequently to Replicat. These anomalies can cause Replicat to abend with duplicate or missing record errors and happen most frequently with IDX hospital applications and some BASE25 bank applications. You can use the Replicat parameter FILTERDUPS to resolve these anomalies.

Selecting rows

To designate 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 253.
- 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. For example, when a filter is part of a conflict resolution configuration, RAISEERROR can be used to trigger a response by means of a REPERROR clause (such as to discard the record).

Text strings within a FILTER clause must be enclosed within double quotes, as shown in the following examples.

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 @STRFIND function to extract records containing a string "JOE."

```
TABLE act.tcustord, FILTER (@STRFIND (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.

(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 37, 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 37 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. Uses the SQLEXEC option to perform 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.

Table 37 Using multiple FILTER statements (continued)

Parameter file	Description

`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 38 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 38 Permissible WHERE operators

Element	Examples
Column names	`PRODUCT_AMT`
Numeric values	`-123, 5500.123`
Literal strings enclosed in quotes	`"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 243.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses ( ) for logical grouping of multiple elements.

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 275).

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.

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:

- Only use primary-key columns as selection criteria, if possible.
- Make required column values available by using 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_src.people, FETCHBEFOREFILTER, FETCHCOLS (age), &
  FILTER (age > 50);
  ```

  It may be more efficient to enable supplemental logging for the required columns than to fetch the values.
- 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:

- 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.

Retrieving before values

For update operations, it can be advantageous to check the *before* values of source columns — the values before the update occurred. Reasons to use before values include the following:

- Comparing the before value of a replicated source column to the current value of the target column (before posting the update) ensures that the operation is being applied to the correct version of the target record. If the values do not match, it could be that the target table is corrupted or was changed by a user or application other than Oracle

GoldenGate. Instead of posting the update, you could configure such comparisons to generate an error and log the operation to an exceptions table so that the problem can be detected and resolved. For example:

```
TABLE hr.people, FETCHBEFOREFILTER, FETCHCOLS (name), &
FILTER (ON UPDATES, BEFORE.name = "Doe");
```

- You can use before values in 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);
```

To reference the before value

1. In the selection clause, prefix the name of the column with the BEFORE keyword followed by a dot (.), as shown below and in the preceding examples:

```
BEFORE.<column_name>
```

2. Use the GETUPDATEBEFORES parameter in the Extract parameter file to extract before images or in the Replicat parameter file to replicate before images. To use this parameter, 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 243.

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)
```

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

Selecting SQL statement types to replicate

By default, Oracle GoldenGate synchronizes 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

Converting one operation type to another

You can convert one type of SQL operation to another in the following manner. All of the following parameters are for use 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.

Mapping columns

Oracle GoldenGate provides for column mapping at the table level and at the global level.

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 column structures to be identical, they must:

- have identical column names (including case, if applicable)
- have identical data types
- have identical column sizes
- have the same column length semantics for character columns (bytes versus characters)
- appear in the same order in each table

For example, a source-definitions file is required when the semantics of a source Oracle database are configured as bytes and the target semantics are configured as characters (or the other way around), even though the table structures may be identical. As another example, a source-definitions file is required for the following set of source and target tables, which are identical except for the order of the name columns:

Source

```
CREATE TABLE emp
( employee_id    NUMBER(6)
, first_name     VARCHAR2(20)
, last_name      VARCHAR2(25)
, phone_number   VARCHAR2(20)
, hire_date      DATE  DEFAULT SYSDATE
```

Target

```
CREATE TABLE emp
( employee_id    NUMBER(6)
, last_name      VARCHAR2(25)
, first_name     VARCHAR2(20)
, phone_number   VARCHAR2(20)
, hire_date      DATE  DEFAULT SYSDATE
```

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.

See "Creating a data-definitions file" on page 115.

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 39 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.
`<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 quotes, such as "ABCD" ◆ The name of a source column, such as ORD_DATE ◆ An expression using an Oracle GoldenGate column-conversion function, such as: @STREXT (COL1, 1, 3)
`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 DEFGEN. 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 250. 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 40 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.

Syntax

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

Table 41 COLMATCH options

Argument	Description

`NAMES <target column> =`
`<source column>`
	Maps based on column names.

Table 41 COLMATCH options (continued)

Argument	Description
`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.

Table 42 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 sales.ord, TARGET sales.order, COLMAP (USEDEFAULTS);
COLMATCH RESET
MAP sales.reg, TARGET sales.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.

- Columns with the same name are mapped to each other if the data types are compatible.
- For databases that do not support case-sensitivity for object names, column names are changed to uppercase for name comparison. For databases that are configured for case-sensitivity, Oracle GoldenGate considers the case when evaluating columns for default mapping.
- 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, then the map is created using the first eligible source column whose name matches that of the target column, regardless of case.
- 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 43.

Table 43 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

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.

Alphanumeric columns

Character-based columns can accept character-based data types such as VARCHAR, GROUP, and date and time types, as well as string literals enclosed in quotes. 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.

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.

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.

To map datetime columns to a Teradata target database that uses a Teradata-ODBC driver version earlier than 3.02.0.02, the following Replicat parameters are available:

USEDATEPREFIX	Prefixes values for DATE data types with a DATE literal.
USETIMEPREFIX	Prefixes values for TIME data types with a TIME literal.
USETIMESTAMPPREFIX	Prefixes values for TIMESTAMP data types with a TIMESTAMP literal.

Handling unprintable characters

Oracle GoldenGate provides the following parameters for handling unprintable characters.

Globally replacing unprintable characters

To correct for unprintable characters globally, use the REPLACEBADCHAR and REPLACEBADNUM parameters in the Replicat parameter file. These parameters specify a value to substitute whenever unprintable characters or numbers are encountered during column mapping. When used, these parameters should be placed before MAP entries in the parameter file.

Controlling spaces-to-null conversion

(Oracle only) When synchronizing source and target tables that have different definitions, Oracle GoldenGate converts columns that contain only spaces to a NULL value in the target table. This behavior is controlled by the SPACESTONULL parameter. To prevent this behavior, use the NOSPACESTONULL parameter, which causes Oracle GoldenGate to write a single space character to the target column.

Trimming trailing spaces in character columns

To control whether or not trailing spaces are truncated when mapping a CHAR column to a VARCHAR column, use the TRIMSPACES and NOTRIMSPACES parameters. These parameters can be used at the root level of a parameter file to turn the trim feature on or off for different TABLE or MAP statements (or groups of statements), and they can be used within individual TABLE or MAP statements to override the global setting. The default is to trim trailing spaces.

Controlling binary data in character columns

Oracle GoldenGate preserves binary characters that are entered into a source or target column that is defined as a character column. A multi-byte character or any type of unprintable binary character is preserved, such as NULL, a carriage return, or a shell command.

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 19 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 19 on page 252, 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.

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 275.

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.
`<argument>`	Function arguments can contain the following:

Syntax element	Description	
	Argument element	**Example**
	A numeric constant	`123`
	A string constant	`"ABCD"` (Must be within quotes.)
	The name of a source column	`PHONE_NO`
	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)`

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.

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 255.

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)
```

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 44 Conversion functions for numbers and characters

Purpose	Conversion Function
Convert a binary or character string to a number.	@NUMBIN @NUMSTR
Convert a number to a string.	@STRNUM
Compare strings.	@STRCMP @STRNCMP
Concatenate strings.	@STRCAT @STRNCAT
Extract from a string.	@STREXT @STRFIND
Return the length of a string.	@STRLEN
Substitute one string for another.	@STRSUB
Convert a string to upper case.	@STRUP
Trim leading or trailing spaces, or both.	@STRLTRIM @STRRTRIM @STRTRIM

Handling null, invalid, and missing data

When column data is missing, invalid, or null, an Oracle GoldenGate conversion function returns a corresponding value.

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

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 alphanumeric character data, either from values obtained from the system, database, transaction, or record, or from values retrieved from queries, procedures, or other called 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 constant that is enclosed within double quotes or the result of an Oracle GoldenGate column-conversion function (see the following example).

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. For more information about @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.

```
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
```

Mapping and transforming Unicode and native characters

Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. An escape sequence can be used in the following elements within a TABLE or MAP statement:

- 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
- \377 Octal escape sequence
- \xFF Hexadecimal escape sequence

The following limitations apply:

- This support is limited to UTF-16 code points from U+0000 to U+007F, the equivalent of 7-bit ASCII.
- The source and target columns must both be Unicode.
- 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.

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)
- This is the only permissible escape sequence to use for NCHAR and NVARCHAR columns.
- Surrogate pairs are not supported.

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.

CHAPTER 18
Customizing Oracle GoldenGate processing

Overview of custom processing

The following features can help you to customize and streamline processing:

- SQL procedures and queries
- Macros
- User exits
- Event markers

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:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types
- BLOB data types

Ingres

All data types except LOB

MySQL

All data types except TEXT and BLOB

Oracle

All Oracle types are supported *except the following*:

- BFILE
- BLOB
- CFILE
- CLOB
- NCLOB
- NTY

SQL Server

- CHAR
- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by Oracle GoldenGate.

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 265).

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.
`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.
`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 265).

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.

Syntax

Parameter syntax	Purpose
`SQLEXEC “exec <procedure name>()”`	Execute a stored procedure
`SQLEXEC “<sql query>”`	Execute a query
`SQLEXEC “<database command>”`	Execute a database command

Argument	Description
`“exec <procedure name> ()”`	Specifies the name of a stored procedure to execute. The statement must be enclosed within double quotes. Example: `SQLEXEC "exec prc_job_count ()"`
`“<sql query>”`	Specifies the name of a query to execute. The query must be contained all on one line and enclosed within double quotes. 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, it might be more efficient to enable 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 45 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 46 Macro definition arguments

Argument	Description
`MACRO`	Required. Indicates an Oracle GoldenGate macro.
`#<macro name>`	The name of the macro. See "Macro naming conventions" on page 270.
`PARAMS (#<p1>, #<p2>)`	The names of parameters. See "Macro naming conventions" on page 270. 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 272.
`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 47 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 273.)

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 270, 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 ggs, PASSWORD ggs123,
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 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.

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*.

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_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.

How you can use the event marker system

Examples of how to use this system would be to start 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
- To activate tracing when a specific account number is detected
- To capture lag history
- To establish a point at which to start batch processes or end-of-day reporting procedures

The event marker feature is supported for the replication of data changes, but not for initial loads.

To use the event marker system

The system requires the following input components:

1. Specify the *event record* that will trigger the action. You can do this by including a FILTER or WHERE clause, or a SQLEXEC query or procedure, in one of the following parameter statements:
    - TABLE statement in an Extract parameter file
    - MAP statement in a Replicat parameter file
    - Special TABLE statement in a Replicat parameter file that enables you to perform EVENTACTIONS actions without mapping a source table to a target table

2. In the same TABLE or MAP statement where you specified 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 about these parameters, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

To combine multiple actions

- Many, but not all EVENTACTIONS options, can be combined. You probably will need to combine two or more actions to achieve your goals.

- The entire EVENTACTIONS statement is parsed first, and only then are the specified options executed according to which one takes precedence over another. In the following list, the actions that are listed before Process the record will occur before the record is written to the trail or applied to the target (depending on the process). Actions that are listed after Process the record will be executed after the record is processed.
  - TRACE
  - LOG
  - CHECKPOINT BEFORE
  - IGNORE
  - DISCARD
  - SHELL
  - ROLLOVER
  - (Process the record)
  - REPORT
  - ABORT
  - CHECKPOINT AFTER
  - FORCESTOP
  - STOP

The following examples show how you can combine EVENTACTIONS options. For more examples, see "Case studies in the usage of the event marker system" on page 285.

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)
```

To control the processing of the event record itself

To prevent the event record itself from being processed in the normal manner, use the IGNORE or DISCARD option. Because IGNORE and DISCARD are evaluated before the record itself, they prevent the record from being processed. Without those options, Extract writes the record to the trail, and Replicat applies the operation that is contained in the record to the target database.

You should take into account the possibility that a transaction could contain two or more records that trigger an event action. In such a case, there could be multiple executions of certain EVENTACTIONS specifications. For example, encountering two qualifying records that trigger two successive ROLLOVER actions will cause Extract to roll over the trail twice, leaving one of the two essentially empty.

Syntax

```
EVENTACTIONS (
[STOP | ABORT | FORCESTOP]
[IGNORE [TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL <command>]
[TRACE <trace file> [TRANSACTION] [PURGE | APPEND]]
[CHECKPOINT [BEFORE | AFTER | BOTH]]
[, ...]
)
```

Action	Description
`STOP`	Brings the process to a graceful stop when the specified event record is encountered. The process waits for open transactions to be completed before stopping. If the transaction is a Replicat grouped or batched transaction, the current group of transactions are applied before the process stops gracefully. The process restarts at the next record after the event record, so long as that record also signified the end of a transaction. The process logs a message if it cannot stop immediately because a transaction is still open. However, if the event record is encountered within a long-running open transaction, there is no warning message that alerts you to the uncommitted state of the transaction. Therefore, the process may remain running for a long time despite the STOP event. STOP can be combined with other EVENTACTIONS options except for ABORT and FORCESTOP.
`ABORT`	Forces the process to exit immediately when the specified event record is encountered, whether or not there are open transactions. The event record is not processed. A fatal error is written to the log, and the event record is written to the discard file if DISCARD is also specified. The process will undergo recovery on startup. ABORT can be combined only with CHECKPOINT BEFORE, DISCARD, SHELL, and REPORT.

Action	Description
`FORCESTOP`	Forces the process to stop gracefully when the specified event record is encountered, but only if the event record is the last operation in the transaction or the only record in the transaction. The record is written normally. If the event record is encountered within a long-running open transaction, the process writes a warning message to the log and exits immediately, as in ABORT. In this case, recovery may be required on startup. If the FORCESTOP action is triggered in the middle of a long-running transaction, the process exits without a warning message. FORCESTOP can be combined with other EVENTACTIONS options except for ABORT, STOP, CHECKPOINT AFTER, and CHECKPOINT BOTH. If used with ROLLOVER, the rollover only occurs if the process stops gracefully.
`IGNORE [TRANSACTION [INCLUDEVENT]]`	By default, forces the process to ignore the specified event record. No warning or message is written to the log, but the Oracle GoldenGate statistics are updated to show that the record was ignored. ◆ Use TRANSACTION to ignore the entire transaction that contains the record that triggered the event. If TRANSACTION is used, the event record must be the first one in the transaction. When ignoring a transaction, the event record is also ignored by default. TRANSACTION can be shortened to TRANS. ◆ Use INCLUDEEVENT with TRANSACTION to propagate the event record to the trail or to the target, but ignore the rest of the associated transaction. IGNORE can be combined with all other EVENTACTIONS options except ABORT and DISCARD.
`DISCARD`	Causes the process to: ◆ write the specified event record to the discard file. ◆ update the Oracle GoldenGate statistics to show that the record was discarded. The process resumes processing with the next record in the trail. When using this option, use the DISCARDFILE parameter to specify the name of the discard file. By default, a discard file is not created. DISCARD can be combined with all other EVENTACTIONS options except IGNORE.

Action	Description
`LOG [INFO | WARNING]`	Causes the process to log the event when the specified event record is encountered. The message is written to the report file, to the Oracle GoldenGate error log, and to the system event log. Use the following options to specify the severity of the message: ◆ INFO specifies a low-severity informational message. This is the default. ◆ WARNING specifies a high-severity warning message. LOG can be combined with all other EVENTACTIONS options except ABORT. If using ABORT, LOG is not needed because ABORT logs a fatal error before the process exits.
`REPORT`	Causes the process to generate a report file when the specified event record is encountered. This is the same as using the SEND command with the REPORT option in GGSCI. The REPORT message occurs after the event record is processed (unless DISCARD, IGNORE, or ABORT are used), so the report data will include the event record. REPORT can be combined with all other EVENTACTIONS options.
`ROLLOVER`	Valid only for Extract. Causes Extract to roll over the trail to a new file when the specified event record is encountered. The ROLLOVER action occurs before Extract writes the event record to the trail file, which causes the record to be the first one in the new file unless DISCARD, IGNORE or ABORT are also used. ROLLOVER can be combined with all other EVENTACTIONS options except ABORT. Note: ROLLOVER cannot be combined with ABORT because: ◆ ROLLOVER does not cause the process to write a checkpoint. ◆ ROLLOVER happens before ABORT. Without a ROLLOVER checkpoint, ABORT causes Extract to go to its previous checkpoint upon restart, which would be in the previous trail file. In effect, this cancels the rollover.

Action	Description
`SHELL <command>`	Causes the process to execute the specified shell command when the specified event record is encountered. ◆ <command> specifies the system or shell command to be issued. ◆ If the shell command is successful, the process writes an informational message to the report file and to the event log. Success is based upon the exit status of the command in accordance with the UNIX shell language. In that language, zero indicates success. ◆ If the system call is not successful, the process abends with a fatal error. In the UNIX shell language, non-zero equals failure. SHELL can be combined with all other EVENTACTIONS options.
`TRACE <trace file>` `[TRANSACTION]` `[PURGE | APPEND]`	Causes process trace information to be written to a trace file when the specified event record is encountered. By default, tracing is enabled until the process terminates. To set the trace level, use the Oracle GoldenGate TRACE or TRACE2 parameter. ◆ <trace file> specifies the name of the trace file and must appear immediately after the TRACE keyword. You can specify a unique trace file, or use the default trace file that is specified with the standalone TRACE or TRACE2 parameter. The same trace file can be used across different TABLE or MAP statements in which EVENTACTIONS TRACE is used. If multiple TABLE or MAP statements specify the same trace file name, but the TRACE options are not used consistently, preference is given to the options in the last resolved TABLE or MAP that contains this trace file. ◆ Use TRANSACTION to enable tracing only until the end of the current transaction, instead of when the process terminates. For Replicat, transaction boundaries are based on the source transaction, not the typical Replicat grouped or batched target transaction. TRANSACTION can be shortened to TRANS. ◆ Use PURGE to truncate the trace file before writing additional trace records, or use APPEND to write new trace records at the end of the existing records. APPEND is the default. TRACE can be combined with all other EVENTACTIONS options except ABORT. To disable tracing to the specified trace file, issue the GGSCI SEND <process> command with the TRACE OFF <filename> option.

Action	Description
`CHECKPOINT [BEFORE | AFTER | BOTH]`	Causes the process to write a checkpoint when the specified event record is encountered. Checkpoint actions provide a context around the processing that is defined in TABLE or MAP statements. This context has a begin point and an end point, thus providing synchronization points for mapping the functions that are performed with SQLEXEC and user exits. ◆ BEFORE BEFORE for an Extract process writes a checkpoint before Extract writes the event record to the trail. BEFORE for a Replicat process writes a checkpoint before Replicat applies the SQL operation that is contained in the record to the target. BEFORE requires the event record to be the first record in a transaction. If it is not the first record, the process will abend. Use BEFORE to ensure that all transactions prior to the one that begins with the event record are committed. CHECKPOINT BEFORE can be combined with all EVENTACTIONS options. ◆ AFTER AFTER for Extract writes a checkpoint after Extract writes the event record to the trail. AFTER for Replicat writes a checkpoint after Replicat applies the SQL operation that is contained in the record to the target. AFTER flags the checkpoint request as an advisory, meaning that the process will only issue a checkpoint at the next practical opportunity. For example, in the case where the event record is one of a multi-record transaction, the checkpoint will take place at the next transaction boundary, in keeping with the Oracle GoldenGate data-integrity model. CHECKPOINT AFTER can be combined with all EVENTACTIONS options except ABORT. ◆ BOTH BOTH combines BEFORE and AFTER. The Extract or Replicat process writes a checkpoint before and after it processes the event record. CHECKPOINT BOTH can be combined with all EVENTACTIONS options except ABORT. CHECKPOINT can be shortened to CP.

Case studies in the usage of the event marker system

Trigger end-of-day processing

This example specifies the capture of operations that are performed on a special table named event_table in the source database. This table exists solely for the purpose of receiving inserts at a predetermined time, for example at 5:00 P.M. every day. When Replicat receives the transaction record for this operation, it stops gracefully to allow operators to start end-of-day processing jobs. By using the insert on the event_table table every day, the operators know that Replicat has applied all committed transactions up to 5:00. IGNORE causes Replicat to ignore the event record itself, because it has no purpose in the target database. LOG INFO causes Replicat to log an informational message about the operation.

Example
```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

Simplify transition from initial load to change synchronization

Event actions and event tables can be used to help with the transition from an initial load to ongoing change replication. For example, suppose an existing, populated source table must be added to the Oracle GoldenGate configuration. This table must be created on the target, and then the two must be synchronized by using an export/import. This example assumes that an event table named source.event_table exists in the source database and is specified in a Replicat TABLE statement.

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.

Example
```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

Stop processing when data anomalies are encountered

This example uses ABORT to stop Replicat immediately with a fatal error if an anomaly is detected in a bank record, where the customer withdraws more money than the account contains. In this case, the source table is mapped to a target table in a Replicat MAP statement for actual replication to the target. A TABLE statement is also used for the source table, so that the ABORT action stops Replicat before it applies the anomaly to the target database. ABORT takes precedence over processing the record.

Example
```
MAP source.account, TARGET target.account;
TABLE source.account, FILTER (withdrawal > balance), EVENTACTIONS (ABORT);
```

Use a heartbeat table and logging to analyze lag

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.

Example

```
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);
```

Trace a specific order number

The following example enables Replicat tracing only for an order transaction that contains an insert operation for a specific order number (order_no = 1). The trace information is written to the order_1.trc trace file. The MAP parameter specifies the mapping of the source table to the target table.

Example

```
MAP sales.order, TARGET rpt.order;

TABLE source.order,
FILTER (@GETENV ("GGHEADER", "OPTYPE") = "INSERT" AND order_no = 1), &
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

Execute a batch process

In this example, a batch process executes once a month to clear the source database of accumulated data. At the beginning of the transaction, typically a batch transaction, a record is written to a special job table to indicate that the batch job is starting. TRANSACTION is used with IGNORE to specify that the entire transaction must be ignored by Extract, because the target system does not need to reflect the deleted records. By ignoring the work on the Extract side, unnecessary trail and network overhead is eliminated.

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.

Example

```
TABLE source.job, FILTER (@streq(job_type = "HOUSEKEEPING")=1), &
EVENTACTIONS (IGNORE TRANSACTION);
```

Propagate only a SQL statement without the resultant operations

This example shows how different EVENTACTIONS clauses can be used in combination on the source and target to replicate just a SQL statement rather than the operations that result from that statement. In this case, it is an INSERT INTO...SELECT transaction. Such a transaction could generate millions of rows that would need to be propagated, but with this method, all that is propagated is the initial SQL statement to reduce trail and network overhead. The SELECTs are all performed on the target. This configuration requires perfectly synchronized source and target tables in order to maintain data integrity.

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.

Example Extract:

```
TABLE source.statement, EVENTACTIONS (IGNORE TRANS INCLUDEEVENT);
```

Replicat:

```
TABLE source.statement, SQLEXEC (<execute SQL statement>), &
EVENTACTIONS (INFO, IGNORE);
```

Committing other transactions before starting a long-running transaction

This use of EVENTACTIONS ensures that all open transactions that are being processed by Replicat get committed to the target before the start of a long running transaction. It forces Replicat to write a checkpoint before beginning work on the large transaction. Forcing a checkpoint constrains any potential recovery to just the long running transaction. Because a Replicat checkpoint implies a commit to the database, it frees any outstanding locks and makes the pending changes visible to other sessions.

Example

```
TABLE source.batch_table, EVENTACTIONS (CHECKPOINT BEFORE);
```

Execute a shell script to validate data

This example executes a shell script that runs another script that validates data after Replicat applies the last transaction in a test run. On the source, an event record is written to an event table named source.event. The record inserts the value "COMPARE" into the event_type column of the event table, and this record gets replicated at the end of the other test data. In the TABLE statement in the Replicat parameter file, the FILTER clause qualifies the record and then triggers the shell script compare_db.sh to run as specified by SHELL in the EVENTACTIONS clause. After that, Replicat stops immediately as specified by FORCESTOP.

Example 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);
```

CHAPTER 19
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 48 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 48 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 20 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 how often to report lag information to the error log. If the lag is greater than 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 21 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 22 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 23 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 24 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

Figure 25 The Oracle GoldenGate Error Log (ggserr.log file)

```
2011-01-08 11:20:56  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).
2011-01-08 11:20:56  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7840.
2011-01-08 11:21:31  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).
2011-01-08 11:21:31  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7841.
2011-01-08 11:24:15  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).
2011-01-08 11:24:15  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7842.
2011-01-08 11:24:16  GGS INFO    399  GoldenGate Command Interpreter for
Oracle:  GGSCI command (ggs): add extract extcust  tranlog, begin now.
2011-01-08 11:30:19  GGS INFO    399  GoldenGate Command Interpreter for
Oracle:  GGSCI command (ggs): add rmttrail /home/ggs, extract ggs
```

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.

Figure 26 Sample Extract process report

```
*********************************************************************
** Running with the following parameters **
*********************************************************************
sourceisfile
userid ggs, password ********
rmthost sys1, mgrport 8040
rmtfile /home/ggsora/dirdat/tcustord.dat, purge
table tcustord;

Processing table TCUSTORD

*********************************************************************
** Run Time Statistics **
*********************************************************************
Report at 2011-01-13 11:07:36 (activity since 2011-01-13 11:07:31)

Output to /home/ggsora/dirdat/tcustord.dat:

From Table TCUSTORD:
       #       inserts: 2
       #       updates:0
       #       deletes:0
       #      discards:0
```

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 27 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.

Figure 28 Sample discard file

```
ORA-20017:  asta0009 6144935
ORA-06512: at "LON.STARTASTA0009_INSERT", line 31
ORA-04088: error during execution of trigger 'LON.STARTASTA0009_INSERT'

Operation failed at seqno 45 rba 12483311
Problem replicating PRODTAB.ASTA0009 to ASTA0009

Error occurred with insert record (target format)...
*
A_TIMESTAMP = 2011-01-15 13:18:32
RELA_PERSON_NR = 3618047
RELA_BEZART = 1
RELA_BEZCODE = 01
RELA_AZ_BAFL = 2819220
RELA_STEMPEL = 0
AKTION = I
OK = 1.0000
NOTOK = -1.0000
*
```

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.

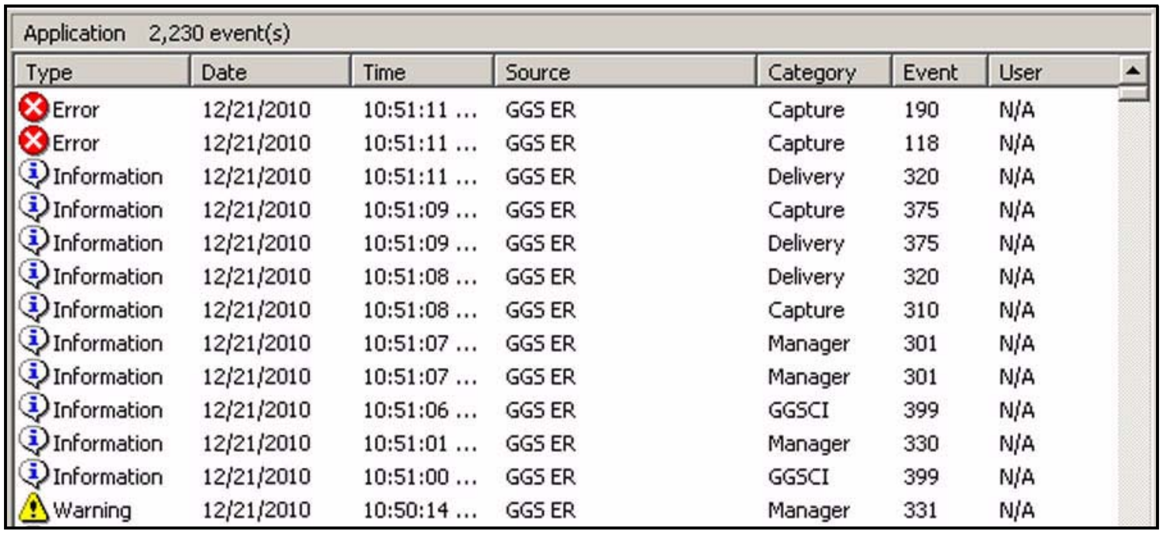
Syntax

```
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.

Figure 29 Oracle GoldenGate messages as seen in the Windows Event Viewer



Application 2,230 event(s)

Type	Date	Time	Source	Category	Event	User
Error	12/21/2010	10:51:11 ...	GGS ER	Capture	190	N/A
Error	12/21/2010	10:51:11 ...	GGS ER	Capture	118	N/A
Information	12/21/2010	10:51:11 ...	GGS ER	Delivery	320	N/A
Information	12/21/2010	10:51:09 ...	GGS ER	Capture	375	N/A
Information	12/21/2010	10:51:09 ...	GGS ER	Delivery	375	N/A
Information	12/21/2010	10:51:08 ...	GGS ER	Delivery	320	N/A
Information	12/21/2010	10:51:08 ...	GGS ER	Capture	310	N/A
Information	12/21/2010	10:51:07 ...	GGS ER	Manager	301	N/A
Information	12/21/2010	10:51:07 ...	GGS ER	Manager	301	N/A
Information	12/21/2010	10:51:06 ...	GGS ER	GGSCI	399	N/A
Information	12/21/2010	10:51:01 ...	GGS ER	Manager	330	N/A
Information	12/21/2010	10:51:00 ...	GGS ER	GGSCI	399	N/A
Warning	12/21/2010	10:50:14 ...	GGS ER	Manager	331	N/A

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 20
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. You can use Oracle GoldenGate to replicate the DDL that a patch or upgrade performs, or you can do those procedures manually on both source and target.

To use Oracle GoldenGate to replicate patch DDL

When you use Oracle GoldenGate to replicate DDL to the target from a patch that is performed on the source, you do not need to stop data replication. To use Oracle GoldenGate for this purpose, see Chapter 14 on page 141. Plan ahead for this, because you will first need to learn about, and configure, the Oracle GoldenGate DDL environment. However, once you have a DDL environment in place, future patches will be much easier to replicate.

Some considerations when using this method:

1. If the application patch or upgrade adds new objects that you want to include in data replication, make certain that you add them to your TABLE and MAP statements. See the procedure for this on page 312.
2. If the application patch or upgrade installs triggers or cascade delete constraints, disable these objects on the target to prevent collisions between the DML that they execute on the target and the same DDL that is replicated from the source trigger or cascaded delete.

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 any definitions of the source and target tables, run the DEFGEN utility for the affected source tables to generate updated source definitions. Replace the old definitions for those tables 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.

   ```
   EDIT PARAMS <group>
   ```

   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>]]
    ```

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

Oracle GoldenGate supports adding tables for synchronization without having to stop and start processes or perform special procedures. The procedure varies, depending on whether or not you used wildcards in the TABLE parameter to specify tables.

To add a table when wildcards are used

When wildcards are used with the TABLE parameter to specify tables for synchronization, Oracle GoldenGate begins synchronizing any new tables whose names satisfy the wildcard pattern. Follow these steps to prepare a new table for synchronization.

1. Stop Extract.

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

2. 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.
3. Add the table name to the TABLE and MAP statements, if needed.
4. Add the new table to the source database and add the target table to the target database. (Oracle GoldenGate does not replicate DDL, so the tables have to be created on both sides.) Do not permit user access to the new table yet.
5. If supported for this database, run the ADD TRANDATA command in GGSCI for the table.
6. If the source and target tables have different definitions, run the DEFGEN utility for the source table to generate source definitions, and then copy the new definitions to the *existing* source definitions file on the target system. If the table you are adding has definitions that match a current definitions template, you do not need to run DEFGEN; just specify the template with the DEF option of the MAP parameter.
7. Permit user access to the table.

To add a table when wildcards are not used

To add a new table to the source database when table names are explicitly defined by their full names (without wildcards), follow these steps.

1. Stop Extract.

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

2. Add the new table to the source database and add the target table to the target database. (Oracle GoldenGate does not replicate DDL, so the tables have to be created on both sides.) Do not permit user access to the new table yet.
3. If supported for this database, run the ADD TRANDATA command in GGSCI for the table.
4. If the source and target tables have different definitions, run the DEFGEN utility for the source table to generate source definitions, and then copy the new definitions to the *existing* source definitions file on the target system. If the table you are adding has definitions that match a current definitions template, you do not need to run DEFGEN; just specify the template with the DEF option of the MAP parameter.
5. Permit user access to the table.

Changing attributes of a source table being synchronized

NOTE See also "Performing an ALTER TABLE to add a column on DB2 z/OS tables"

To add or change columns or partitions, to change supplemental logging details (Oracle), or to make other changes to a table that already is being synchronized, you must stop and start the Oracle GoldenGate processes so that the new attributes can be added to the object record.

1. Stop user and application access to the table being changed.
2. Make the change to the source table.
3. Take note of the current time.

4. In GGSCI, issue the INFO EXTRACT command with SHOWCH until you see that the current checkpoint of Extract has moved past the time that you noted.

   ```
   INFO EXTRACT <group name>, SHOWCH
   ```

5. Stop Extract.

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

6. Issue the INFO REPLICAT command until you see that Replicat has reached the end of the trail (EOF).
7. Make the change to the target table.
8. If the source and target tables have different definitions, run the DEFGEN utility for the source table to generate updated source definitions, and then replace the old definitions for that table with the new ones in the *existing* source definitions file on the target system.
9. Start Extract and then start Replicat.

   ```
   START EXTRACT <group name>
   START REPLICAT <group name>
   ```

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 Extract group.

   ```
   DELETE EXTRACT <group name>
   ```

4. Using standard operating system commands, remove the trail files.
5. Add the Extract group again, specifying the new number of threads.

   ```
   ADD EXTRACT <group name> TRANLOG, THREADS <n>, BEGIN NOW
   ```

6. Add the trail again.

   ```
   ADD RMTTRAIL <trail name>, EXTRACT <group name>
   ```

7. Open the Extract parameter file and change the RMTTRAIL parameter to point to the new trail.

   ```
   EDIT PARAMS <group name>
   ```

8. Start Extract.

```
START EXTRACT <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 21
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.

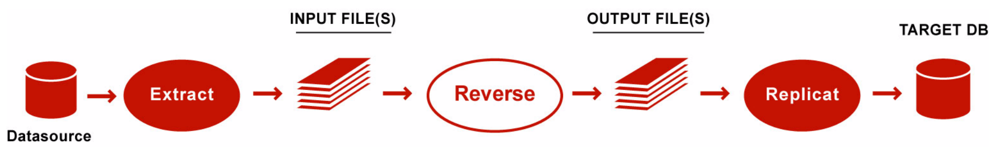
To use the Reverse utility, you do the following:

- 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 30 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 49 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

You can extract transaction data to either of the following:

- As a batch process, created and started from the command shell, that writes to one extract file or to a series of extract files. When multiple files are used, Oracle GoldenGate automatically reverses the file order during reverse processing so that transaction sequencing is maintained. For example:

  File IN000004 gets reversed and written to OUT000001.

  File IN000003 gets reversed and written to OUT000002.

  File IN000002 gets reversed and written to OUT000003.

  ... and so forth.
- An online process, created and started through GGSCI, that writes to a standard local or remote trail. 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 50 and Table 51. In addition to these parameters, include any other optional parameters or special MAP statements that are required for your synchronization configuration.

Table 50 Extract parameter file for the Reverse utility

Parameter	Description
`{EXTRACT <group> | SPECIALRUN, TRANLOG}`	◆ EXTRACT <group> specifies that this will be an online Extract process. You will create this process in GGSCI. ◆ SPECIALRUN, TRANLOG specifies that this is a special batch run and defines the transaction log as the source for extracting the before data.
`BEGIN <time>`	(SPECIALRUN only) The timestamp in the data source at which to begin the reverse processing. Processing starts with the first record that has a timestamp greater than, or equal to, the time specified with BEGIN. Note: To specify a begin time for an online process, you will use the ADD EXTRACT command.
`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>]]`
◆ 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 ggs123`
	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.

Table 50 Extract parameter file for the Reverse utility (continued)

Parameter	Description
`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.

`{EXTFILE <input file>|`
`RMTFILE <input file>}`
`[, MAXFILES <n>]`
 	
`or...`
 	
`{EXTTRAIL <input trail> |`
`RMTTRAIL <input trail>}`
	Can be one of the following to specify the *input* file(s) to the Reverse utility.
	If using SPECIALRUN for a batch run:
	◆ Use EXTFILE to specify an extract file on the local system.
	◆ Use RMTFILE to specify an extract file on a remote system.
	Specify the relative or full path name of a file, for example:
	`EXTFILE /home/ggs/dirdat/input.dat`
	To create a series of files that roll over similar to an Oracle GoldenGate trail, use the MAXFILES option. This option can be used to accommodate file-size limits of the operating system.
	If using GGSCI to create an online Extract group:
	◆ 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 <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 extract file.

```
SPECIALRUN, TRANLOG
BEGIN 2011-01-09 14:04:55
END 2011-01-09 14:12:20
USERID ggs, PASSWORD ggs
GETUPDATEBEFORES
NOCOMPRESSDELETES
RMTHOST sysb, MGRPORT 8040
RMTFILE /home/ggs/dirdat/input.dat, purge
TABLE tcustmer;
TABLE tcustord;
```

Example This example Extract parameter file uses a series of remote extract files that roll over as needed until a maximum of 10 files is on disk.

```
SPECIALRUN, TRANLOG
BEGIN 2011-01-09 14:04:55
END 2011-01-09 14:12:20
USERID ggs, PASSWORD ggs
GETUPDATEBEFORES
NOCOMPRESSDELETES
RMTHOST sysb, MGRPORT 8040
RMTFILE /home/ggs/dirdat/in, MAXFILES 10
TABLE tcustmer;
TABLE tcustord;
```

Example This example Extract parameter file uses a remote trail.

```
EXTRACT ext_1
END 2011-01-09 14:12:20
USERID ggs, PASSWORD ggs
GETUPDATEBEFORES
NOCOMPRESSDELETES
RMTHOST sysb, MGRPORT 8040
RMTTRAIL /home/ggs/dirdat/in
TABLE tcustmer;
TABLE tcustord;
```

Table 51 Replicat parameter file for the Reverse utility

Parameter	Description
`{REPLICAT <group> | SPECIALRUN}`	◆ REPLICAT <group> specifies that this will be an online Replicat process to be created in GGSCI. ◆ SPECIALRUN specifies that this is a special batch run Replicat process.
`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 51 Replicat parameter file for the Reverse utility (continued)

Parameter	Description

`EXTFILE <output file>`
`[, MAXFILES <n>]`
	Use only if using SPECIALRUN. Specifies *output* file(s) for Reverse utility. Replicat reads from the output file(s) to apply the before data to the database, restoring it to the previous state.
	This file must be different from the input file that was specified with EXTFILE or RMTFILE. Use a unique, relative or fully qualified name, for example:
	`EXTFILE /home/ggs/dirdat/output.dat`
	To create a series of files that rolls over similar to an Oracle GoldenGate trail, use the MAXFILES option. This option can be used to accommodate file-size limits of the operating system.

`[TARGETDB <dsn>,]`
`[USERID <user id>`
`[, PASSWORD <password>]]`
◆ 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 ggs123`
	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 115 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 using SPECIALRUN.

```
SPECIALRUN
END RUNTIME
EXTFILE /home/ggs/dirdat/output.dat
USERID ggs, PASSWORD ggs
ASSUMETARGETDEFS
MAP tcustmer, TARGET tcustmer;
```

Example The following is an example Replicat parameter file using an online Replicat group.

```
REPLICAT rep_1
END RUNTIME
USERID ggs, PASSWORD ggs
ASSUMETARGETDEFS
MAP tcustmer, TARGET tcustmer;
```

Creating online process groups and trails for reverse processing

To use online process groups to perform the backout procedure, you must 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 online 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 10).

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 121).
- 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 10).

Running the Reverse utility

To perform the reverse processing as a batch job

Follow these steps within the operating system's command shell. When running the programs, use the full path name of the Oracle GoldenGate directory.

1. From the command shell, run Extract with the Extract parameter file that you created. Extract will capture the data that is specified in the Extract parameter file.

```
/<GoldenGate_directory>/extract paramfile irprm/<Extract_paramfile>.prm
```

2. When the data extraction is complete, 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 EXTFILE or RMTFILE in the Extract parameter file. Use a wildcard to specify multiple files in a series, for example in*.
- <output file> is the output file specified with EXTFILE in the Replicat parameter file. Use a wildcard to specify multiple files in a series, for example out*.

WARNING On a UNIX system, do *not* put a space after the file name. Otherwise, the UNIX file system will not pass file names properly back to Oracle GoldenGate.

Example
```
/home/ggs/reverse input.dat, output.dat
```

Example
```
/home/ggs/reverse in*, out*
```

3. When reverse is finished running, run Replicat with the Replicat parameter file that you created. This applies the reversed-out data to the database.

```
/<GoldenGate_directory>/replicat paramfile dirprm/<Replicat param
file>.prm
```

To perform the reverse processing as an online process

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

Oracle GoldenGate record format

This appendix describes the format of Oracle GoldenGate records that are written to a trail or extract file. 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.

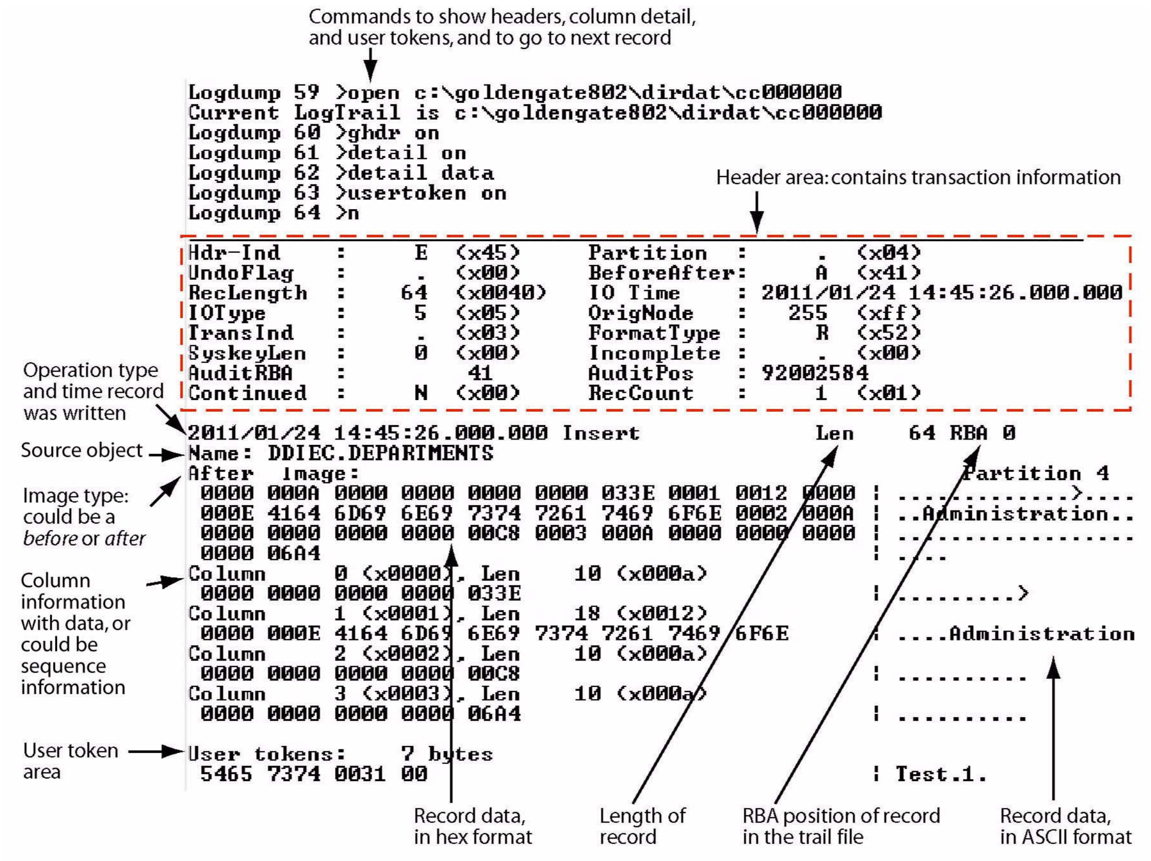
Oracle GoldenGate trail files are unstructured. You can view Oracle GoldenGate records with the Logdump utility provided with the Oracle GoldenGate software. For more information, see the Logdump documentation in the *Windows and UNIX Troubleshooting and Tuning Guide*.

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 31 Sample trail record as viewed with the Logdump utility



```
Logdump 59 >open c:\goldengate802\dirdat\cc000000
Current LogTrail is c:\goldengate802\dirdat\cc000000
Logdump 60 >ghdr on
Logdump 61 >detail on
Logdump 62 >detail data
Logdump 63 >usertoken on
Logdump 64 >n

Hdr-Ind     :     E  (x45)     Partition  :     .  (x04)
UndoFlag    :     .  (x00)     BeforeAfter:     A  (x41)
RecLength   :    64  (x0040)   IO Time    : 2011/01/24 14:45:26.000.000
IOType      :     5  (x05)     OrigNode   :   255  (xff)
TransInd    :     .  (x03)     FormatType :     R  (x52)
SyskeyLen   :     0  (x00)     Incomplete :     .  (x00)
AuditRBA    :          41      AuditPos   : 92002584
Continued   :     N  (x00)     RecCount   :     1  (x01)

2011/01/24 14:45:26.000.000 Insert                Len    64 RBA 0
Name: DDIEC.DEPARTMENTS
After  Image:                                              Partition 4
 0000 000A 0000 0000 0000 0000 033E 0001 0012 0000 | ...............>....
 000E 4164 6D69 6E69 7374 7261 7469 6F6E 0002 000A | ..Administration..
 0000 0000 0000 0000 00C8 0003 000A 0000 0000 0000 | ....................
 0000 06A4                                         | ....
Column     0 (x0000), Len    10 (x000a)
 0000 0000 0000 0000 033E                          | .........>
Column     1 (x0001), Len    18 (x0012)
 0000 000E 4164 6D69 6E69 7374 7261 7469 6F6E      | ....Administration
Column     2 (x0002), Len    10 (x000a)
 0000 0000 0000 0000 00C8                          | ..........
Column     3 (x0003), Len    10 (x000a)
 0000 0000 0000 0000 06A4                          | ..........

User tokens:    7 bytes
 5465 7374 0031 00                                 | Test.1.
```

Commands to show headers, column detail, and user tokens, and to go to next record
Header area: contains transaction information
Operation type and time record was written
Source object
Image type: could be a *before* or *after*
Column information with data, or could be sequence information
User token area
Record data, in hex format
Length of record
RBA position of record in the trail file
Record data, in ASCII format

Record header area

The Oracle GoldenGate record header provides metadata of the data that is contained in the record and includes the following information.

- The operation type, such as an insert, update, or delete
- The before or after indicator for updates
- Transaction information, such as the transaction group and commit timestamp

Description of header fields

The following describes the fields of the Oracle GoldenGate record header. Some fields apply only to certain platforms.

Table 52 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 53 on page 334 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 52 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 52 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

For more information about the @GETENV function, see the *Windows and UNIX Reference Guide*.

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 53 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 53 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 53 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. ◆ ARTYPE_FILECLOSE_GGS 201 — the source application closed a file that was open for unstructured I/O. Used by Replicat.	NSK non-TMF

Table 53 Oracle GoldenGate operation types (continued)

Type	Description	Platform
	◆ 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*.

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.
batch run	A one-time processing run that has a distinct beginning and an end, as opposed to continuous processing that does not have a specific end point, such as online change synchronization.
before image	The values that exist in a row in a database before a SQL operation is performed on that row.

Term	Definition
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*.
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.

Term	Definition
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."
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.

Term	Definition
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
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.

Term	Definition
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.
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 a batch run or 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.

Term	Definition
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*.
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.

Term	Definition
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.
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.
special run	See *batch run*.
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.

Term	Definition
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.
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 batch run 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.

Term	Definition
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.
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